The Many Faces of Planarity

– Matching, Augmentation, and Embedding Algorithms for Planar Graphs –

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation
von

Ignaz Rutter
aus Karlsruhe

Tag der mündlichen Prüfung: 1. Juli 2011

Erster Gutachter: Frau Prof. Dr. Dorothea Wagner

Zweiter Gutachter: Herr Prof. Dr. Alexander Wolff

Acknowledgments

This thesis is the result of roughly four years of work. During this phase, I got support from various sides, and I would like to take the opportunity to thank everyone who supported me during this time.

First and foremost, I thank my two advisors Dorothea Wagner and Alexander (Sascha) Wolff for offering me the chance to do a PhD with them and all the encouragement and support I received during this time. I appreciate especially their trusting and benevolent style of leadership. By first advising my diploma thesis and then offering me the final year of funding in his DFG project "GeoNet" Sascha introduced me to the wonderful world of science and guided me during my first steps there. Although we never had positions at the same place, his door was always open for me, and I have tremendously gained from our frequent research meetings during his time in Eindhoven and afterwards. After my first year, Dorothea completely adopted me into her great group; the productive atmosphere, shaped by mutual trust, encouragement and the freedom to work on almost anything is legendary. In particular, I thank both my advisors for their competent and extensive feedback on my work and for their constant encouragement to attend workshops, conferences and summer schools. I am glad that this thesis is not the end of this collaboration.

Special thanks go to my office mates (in chronological order) Martin Nöllenburg, Thomas Pajor, and Sascha Meinert. You were often the first to turn to for asking question and discussing ideas. Let me extend these thanks to all my colleagues, Reinhard Bauer, Michael Baur, Daniel Delling, Julian Dibbelt, Marco Gaertler, Andreas Gemsa, Robert Görke, Tanja Hartmann, Martin Holzer, Bastian Katz, Marcus Krug, Steffen Mecke, Sascha Meinert, Martin Nöllenburg, Thomas Pajor, Andrea Schumm, and Markus Völker. In particular, I thank Marcus Krug and Bastian Katz for countless hours of discussions on various problems and Andreas Gemsa and Martin Nöllenburg for working with me on interesting research problems, when I desperately needed a distraction from finishing the write-up of this thesis. Moreover, I thank our secretaries Lilian Beckert and Elke Sauer, who helped me to easily take all bureaucratic hurdles, and our system administrator Bernd Giesinger, who keeps our systems up and running and our data safe.

I also thank my numerous coauthors, Patrizio Angelini, Giuseppe Di Battista, Mark de Berg, Thomas Bläsius, Edith Brunel, Robert Franke, Fabrizio Frati, Andreas Gemsa, Dirk Gerrits, Emilio Di Giacomo, Luca Grilli, Vít Jelínek, Mong-Jen Kao, Bastian Katz, Amirali Khosravi, Marcus Krug, Jan Kratochvíl, D.T. Lee, Giuseppe Liotta, Andreas Lochbihler, Martin Nöllenburg, Thomas Pajor, Maurizio Patrignani, Gregor Snelting, Ben Strasser, Constantinos Tsirogiannis, Dorothea Wagner, Gerhard Woeginger, and Alexander Wolff. Working with you has been a great pleasure and I am looking forward to working with many of you in the future.

Special thanks go to Walter Didimo and Beppe Liotta for inviting me to the Bertinoro Workshop on Graph Drawing in 2009, 2010 and 2011. Especially the 2009 edition has

been very fruitful for me and has greatly influenced this thesis. I would also like to thank Giuseppe Di Battista for inviting me to Rome for a research stay, and Jan Kratochvíl for inviting me to Prague twice.

Last but not least I thank my wife Eva for her constant support, encouragement and patience, in particular in the final weeks before handing in this thesis.

Deutsche Zusammenfassung

Ein Graph ist planar, wenn er sich kreuzungsfrei in die Ebene zeichnen lässt. Planarität ist eine zentrale Eigenschaft, nicht nur im Graphenzeichnen, sondern in der gesamten Graphentheorie. Oftmals lassen sich für planare Graphen stärkere theoretische Aussagen beweisen und effizientere Algorithmen angeben als für allgemeine Graphen. Andererseits tritt Planarität oft auch als Nebenbedingung auf und macht Probleme dadurch schwieriger. Eine besondere Rolle spielen planare Graphen in der Visualisierung, da Kreuzungen die Lesbarkeit von Zeichnungen verschlechtern.

Kuratowski [Kur30] legte 1930 den Grundstein für die systematische Untersuchung der planaren Graphen durch eine vollständige Charakterisierung mittels verbotener Substrukturen. Diese Charakterisierung über verbotene Substrukturen, nämlich K_5 und $K_{3,3}$, zeigt, dass Planarität ein „endliches“ Problem ist und führte zu den ersten polynomiellen Erkennungsalgorithmen. Den ersten Linearzeitalgorithmus zur Erkennung von planaren Graphen veröffentlichten Hopcroft und Tarjan erschien 1974 [HT74]. Seitdem ist eine Fülle von Resultaten über planare Graphen erschienen. Beispielsweise besitzen planare Graphen gute Zerlegungseigenschaften, lassen sich mit wenigen Farben färben und viele Lösungen für Standardprobleme, die als Subroutine in anderen Algorithmen eingesetzt werden, lassen sich auf planaren Graphen sehr effizient implementieren. Hierzu zählen beispielsweise Matching- und Flussalgorithmen. Zudem dienen planare Graphen oft als Sprungbrett für die Entwicklung effizienter Algorithmen auf allgemeineren Graphklassen, etwa Graphen mit beschränktem Genus oder den sogenannten H-minorenfreien Graphen.

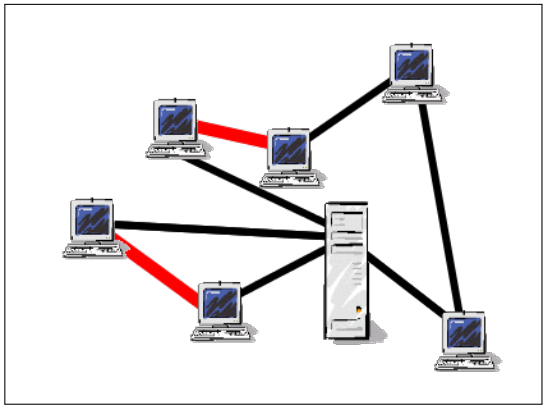
Diese Arbeit liefert einen Beitrag zur Theorie der planaren Graphen. Neben der grundlegenden Klassifikation in polynomiell lösbare und NP-schwere Probleme liegt der Schwerpunkt vor allem auf effizienten Lösungsverfahren mit möglichst linearer Laufzeit. Hierzu werden die Probleme zunächst hinsichtlich ihrer kombinatorischen Eigenschaften untersucht und – sofern sie nicht NP-schwer sind – möglichst einfache Algorithmen angegeben, die die polynomielle Lösbarkeit belegen. Anschließend werden diese Algorithmen schrittweise verfeinert. In vielen Fällen wird dadurch optimale lineare Laufzeit erreicht.

Die Arbeit ist in zwei Teile gegliedert. Im ersten Teil stehen Fragestellungen der kombinatorischen Optimierung im Vordergrund. Dort treten zwei Facetten von Planarität zutage: Einerseits wird Planarität als zusätzliche, hilfreiche Eigenschaft der Eingabe ausgenutzt, andererseits tritt Planarität auch als Nebenbedingung auf, deren Einhaltung durch die Problemstellung gefordert wird und die Probleme häufig schwieriger macht. Der zweite Teil befasst sich mit dem Zeichnen von planaren Graphen. Dabei spielt die Wahl der Einbettung eines planaren Graphen eine wesentliche Rolle für die Qualität der Darstellung. Es wird eine Reihe von Verfahren vorgestellt, die möglichst gute Einbettungen von planaren Graphen für verschiedene Zeichenstile berechnen.

Kombinatorische Optimierung auf planaren Graphen

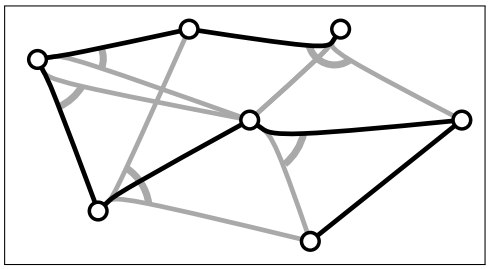
Dieser Teil der Arbeit beschäftigt sich mit verschiedenen kombinatorischen Optimierungsproblemen auf planaren Graphen. Dabei werden drei Fragestellungen behandelt.

Graphaugmentierung Oft möchte man einen gegebenen Graphen durch Hinzufügen möglichst weniger Kanten so verändern, dass er eine gewissen Eigenschaft erhält. In diesem Abschnitt wird eine Reihe von solchen *Graphaugmentierungs-Problemen* betrachtet, bei denen der Zusammenhangsgrad eines Graphen erhöht werden soll. Das Einfügen der dicken roten Kanten in der Abbildung rechts sichert das Netzwerk gegen den Ausfall einer einzelnen Kante ab. In dieser Arbeit wird zusätzlich gefordert, dass der augmentierte Graph planar bleibt. Dieses Problem tritt beispielsweise im Graphenzeichnen auf, da viele Zeichenalgorithmen für planare Graphen zweifachen Zusammenhang voraussetzen oder zumindest für diese Art von Graphen besondere Qualitätsgarantien angeben. Die Forderung möglichst wenige Kanten hinzuzufügen sorgt dafür, dass diese Qualitätsgarantien möglichst gut erhalten bleiben.



Es wird der Komplexitätsstatus einer Reihe von planaren Augmentierungsproblemen untersucht. Insbesondere stellt sich heraus, dass diese Probleme NP-schwer sind, sowohl für den gewöhnlichen Planaritätsbegriff, als auch wenn der Graph geometrisch eingebettet ist, also jeder Knoten bereits eine fest zugewiesene Position hat und die Kanten geradlinig gezeichnet werden müssen. Andererseits wird gezeigt, dass sich eine Reihe von Spezialfällen dennoch effizient lösen lässt, etwa das Absichern einer Verbindung zwischen zwei festen Knoten.

Schaltergraphen Schaltergraphen bieten erweiterte Modellierungsmöglichkeiten gegenüber gewöhnlichen Graphen. Ein Schalter besteht aus einer Menge von Kanten, die sich einen gemeinsamen Knoten teilen. Eine Konfiguration wählt aus jedem Schalter eine Kante aus. Ein Schaltergraph beschreibt also eine Familie von Graphen und eine Konfiguration beschreibt ein konkretes Mitglied dieser Familie. Die nebenstehende Abbildung zeigt einen Schaltergraphen, wobei Kanten, die zum selben Schalter gehören, durch einen Bogen am gemeinsamen Knoten miteinander verbunden sind. Die hervorgehobenen Kanten bilden eine Konfiguration dieses Schaltergraphen, deren resultierender Graph zusammenhängend ist. Aufgrund ihres Aufbaus eignen sich Schaltergraphen gut, um graphentheoretische Probleme zu modellieren, die strukturelle Entscheidungen beinhalten. Hat man einen Schaltergraphen vorliegen, ist man daran interessiert herauszufinden, ob seine Familie einen Graphen enthält, der eine gegebene Grapheigenschaft besitzt.



Es wird gezeigt, dass dieses Problem für die Eigenschaften eulersch, bipartit und planar NP-schwer ist. Andererseits werden effiziente Algorithmen für Zusammenhang und Pfade zwischen vorgegebenen Knoten angegeben.

Große Matchings in planaren Graphen Ein Matching ist eine Teilmenge der Kanten eines Graphen bei der jeder Knoten zu höchstens einer Kante dieser Menge inzident ist. Das Finden von möglichst großen Matchings ist ein gut untersuchtes Problem aus der kombinatorischen Optimierung. Nishizeki und Baybars [NB79] zeigten, dass in planaren Graphen mit festem Minimalgrad stets Matchings einer bestimmten Mindestgröße existieren. Dennoch ist unklar, ob ein solches Matching, von dem man ja weiß, dass es existiert, schneller gefunden werden kann als ein größtes Matching.

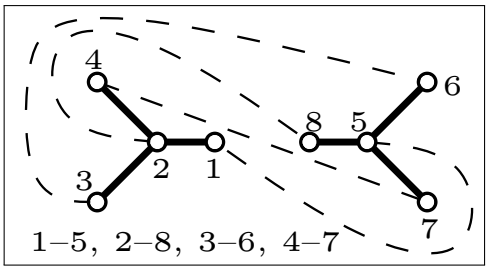
In dieser Arbeit wird gezeigt, wie fester Minimalgrad in planaren Graphen eingesetzt werden kann, um Matchings mit garantierter Mindestgröße schnell zu berechnen. Für Minimalgrad 3 wird dabei die scharfe Schranke von Nishizeki und Baybars erreicht.

Einbettungen von planaren Graphen

Der zweite Teil der Arbeit beschäftigt sich mit der Problemstellung, Einbettungen von planaren Graphen zu finden, die möglichst gut für bestimmte Visualisierungsarten geeignet sind.

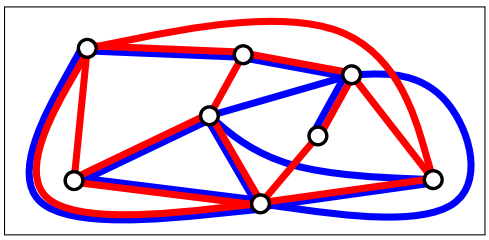
Diese Art von Problemen ist inhärent schwierig, da planare Graphen im Allgemeinen exponentiell viele planare Einbettungen besitzen. Es werden Einbettungsprobleme für unterschiedliche Zeichenstile untersucht, sowohl für topologische Zeichnungen, bei denen Kanten als beliebige Kurven gezeichnet werden dürfen, als auch für orthogonale Zeichnungen, bei denen Kanten nur aus horizontalen und vertikalen Streckensegmenten zusammengesetzt werden.

Planarität partiell eingebetteter Graphen In einer *topologischen Zeichnung* eines Graphen werden Knoten durch Punkte und Kanten durch Jordankurven zwischen ihren Endpunkten repräsentiert. Dieser Teil der Arbeit beschäftigt sich mit der Frage der Erweiterbarkeit planarer Zeichnungen. Gegeben ein Graph G sowie eine topologische Zeichnung eines Teilgraphen H von G stellt sich folgende Frage: Kann die gegebene Zeichnung zu einer planaren Zeichnung von G erweitert werden, ohne die Zeichnung von H zu verändern? Die nebenstehende Abbildung zeigt eine Teilzeichnung eines Graphen (fett gezeichnet) und die planare Ergänzung um die angegebenen Kanten (gestrichelt). Es ist nicht möglich zusätzlich die Kante 1–8 auf planare Art und Weise einzufügen ohne die vorgegebene Teilzeichnung zu verändern, obwohl der resultierende Graph noch planar ist. Für geradliniges Zeichnen ist bekannt, dass das Zeichnungs-Erweiterungsproblem NP-schwer ist [Pat06].



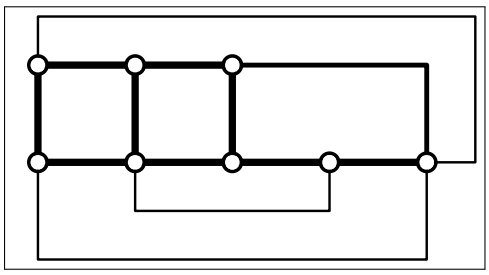
In dieser Arbeit wird ein Algorithmus angegeben, der das Problem für topologische Zeichnungen in Linearzeit löst. Dies ist insofern bemerkenswert, als es sich um ein typisches Lösungs-Erweiterungsproblem handelt und diese oftmals bedeutend schwieriger sind als die entsprechende Problemstellung ohne vorgegebene Teillösung. Weiterhin werden die lösbaren Instanzen, also die Instanzen, die sich planar erweitern lassen, nach dem Vorbild des berühmten Satzes von Kuratowski für planare Graphen [Kur30], durch verbotene Substrukturen charakterisiert. Dabei ergibt sich zusätzlich ein effizienter, zertifizierender Algorithmus für das Einbettungs-Erweiterungsproblem, der im Falle der Lösbarkeit eine entsprechende Einbettung findet und anderenfalls eine verbotene Substruktur liefert.

Simultane Einbettungen Liegen zwei (oder mehr) Graphen auf derselben Knotenmenge vor, so ist man häufig daran interessiert, diese Graphen miteinander zu vergleichen, beispielsweise durch Angabe einer Zeichnung, die die Ähnlichkeiten möglichst gut hervorhebt. Selbst für Paare planarer Graphen ist der Vereinigungsgraph im Allgemeinen nicht planar. Daher sucht man nach einer sogenannten *simultanen Einbettung mit festen Kanten*, das heißt die Knoten der beiden Graphen werden an dieselben Positionen gezeichnet, gemeinsame Kanten werden durch dieselbe Kurve dargestellt und jede Zeichnung für sich genommen ist planar (siehe Beispiel rechts). Obwohl sich in den letzten Jahren viele Wissenschaftler mit diesem Problem beschäftigt haben, ist der Komplexitätsstatus des Problems für Paare allgemeiner planarer Graphen noch ungeklärt.



In dieser Arbeit werden einige Spezialfälle behandelt, unter anderem wird für den Fall, dass der Durchschnitt beider Graphen zweifach zusammenhängend ist, ein polynomieller Algorithmus angegeben. Durch geschickte Anwendung von dynamischer Programmierung wird der zunächst kubische Algorithmus auf lineare Laufzeit beschleunigt.

Orthogonale Zeichnungen mit Flexibilitäts-Bedingungen In einer *orthogonalen* Zeichnung eines Graphen sind alle Kanten aus horizontalen und vertikalen Streckensegmenten zusammengesetzt. Da Kanten mit vielen Knicken die Lesbarkeit der Zeichnungen verschlechtern, ist man bestrebt, die Anzahl der Knicke in der Zeichnung möglichst klein zu halten. Klassischerweise versucht man entweder die Gesamtanzahl an Knicken oder die maximale Anzahl an Knicken pro Kante zu minimieren. Beide Probleme sind NP-schwer, da es NP-schwer ist zu entscheiden, ob sich ein gegebener Graph ganz ohne Knicke zeichnen lässt [GT01]. Es ist aber bekannt, dass sich bis auf eine einzige Ausnahme alle planaren Graphen mit höchstens zwei Knicken pro Kante zeichnen lassen [BK94]. Dagegen war bisher unklar, ob man effizient entscheiden kann, ob sich ein Graph mit nur einem Knick pro Kante orthogonal zeichnen lässt.



In dieser Arbeit wird ein neues Problem dieser Art eingeföhrt, bei dem jeder Kante eine Maximalzahl von Knicken zugeordnet wird, ihre *Flexibilität*. Dies umfasst das Problem der Ein-Knick-Zeichenbarkeit, ist aber noch wesentlich allgemeiner. In der nebenstehenden Abbildung ist die Wichtigkeit der Kanten durch ihre Dicke angegeben. Da wichtige Kanten weniger Knicke haben, ist die Zeichnung besonders gut lesbar. Erlaubt man auch starre Kanten, also solche mit Flexibilität 0, so ist das Problem NP-schwer. Es wird gezeigt, dass das Problem effizient lösbar ist, sofern man jeder Kante mindestens einen Knick erlaubt. Durch geschickte Wiederverwendung von Teilergebnissen, lässt sich der resultierende Algorithmus mit quadratischer Laufzeit implementieren.

Contents

Chapter 1

Introduction

The foundations to graph theory were laid in the 18th century when Leonhard Euler resolved the problem of the Seven Bridges of Königsberg, which asked whether it was possible to take a walk through Königsberg that uses each of the seven bridges, each crossing the river Pregel, exactly once; see Figure 1.1. Euler's ingenious insight was that the choice of routes inside each landmass, and the exact location at which the bridges attach to landmasses are irrelevant for solving the problem. In fact, these two insights led to the development of two very fruitful fields in mathematics, namely *topology* and *graph theory*. After applying these two ideas to the problem of the seven bridges, what remains is a list of landmasses together with the information which landmasses are connected by which bridge. This is exactly a graph in modern terminology, a term that was introduced only more than a hundred years later, in 1878, by Sylvester [Syl78], who studied chemical bonds.

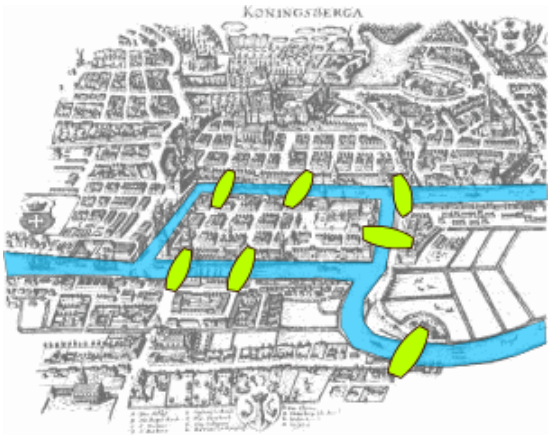


Figure 1.1.: The seven bridges of Königsberg.

It is worth noting that even this first instance of a graph was a planar graph, and actually one with a given planar embedding inherited from the layout of the landmasses and bridges in the city. Although planarity does not play a role in Euler's solution to the problem, it was probably helpful that the "graph" could be studied by looking at its drawing in the form of a map. Sylvester, who introduced the notion of a graph, studied chemical bonds, which also have a natural embedding, albeit in space and not in the plane. These observations make clear that embeddings of graphs into topological spaces and in particular into the plane are not artificial concepts but they where present even at the starting point of graph theory and played an important role in its development. Another indication for this inter-connection is one of the early graph-theoretic problems that spawned a lot of research and results; the four-color problem. In 1852, Francis Guthrie posed the question whether it was always possible to color the regions of a planar map with four colors so that no two regions that share a boundary have the same color.

Today, it can be said without exaggeration that graph theory is one of the big success stories of discrete mathematics and, as it comes to automated processing, also of computer science. Graphs are ubiquitous; they are used in areas well beyond these two fields to model, study and understand various relations among very different entities, for example,

in physics, biology and social sciences, but also to describe IT infrastructure networks or process models. Humans are very visually oriented. Hence, the usage of graphs for describing complex relations often goes together with a corresponding visualization of graphs. If edges cross in a drawing, readability degrades quickly. Therefore, it is intuitive to avoid crossings altogether. This leads to the question which graphs are drawable without any crossings at all, and thus to the definition of the class of planar graphs.

Nowadays, planarity is one of the central concepts not only in graph drawing, but in graph theory as a whole. The characterization of planar graphs proved by Kuratowski [Kur30] in 1930 marks the beginning of modern graph theory. Such a characterization, based on two forbidden topological subgraphs – K_5 and $K_{3,3}$ – makes planarity a finite problem and leads to a polynomial-time recognition algorithm. Planarity is thus "simple" from a computational point of view (this, of course, does not mean that algorithms for testing planarity are trivial) in the strongest possible way, as several linear-time algorithms for testing planarity are known [BM04, HT74, dFdMR06].

Planar graphs constitute the class of graphs that is probably best studied. A rich body of literature shows the tremendous interest in their properties, drawing algorithms, and optimization algorithms that are custom-tailored for planar instances. Nowadays, there is a plethora of results on planar graphs available, and it is well beyond the scope of this thesis to give a comprehensive overview of what is known about planar graphs. We wish to survey, however, at least the most useful properties and available tools for the treatment of planar graphs.

The most basic properties are that planar graphs have only linearly many edges, and they always contain a vertex of degree at most 5. Many properties of planar graphs have immediate algorithmic consequences. For example, the fact that every planar graph has a vertex of low degree immediately implies algorithms for quickly coloring planar graphs with few colors, and also a data structure of linear size that allows to check in constant time, whether two given vertices are adjacent. Another useful property is that planar graphs have good decomposition properties; they have small separators that split the graph in a relatively balanced manner [LT79], which is particularly important for divide & conquer based algorithms. Among others this has lead to a linear-time algorithm for computing shortest paths in planar graphs [HKRS97]. A frequently used property is that planar graphs admit the notion of a dual graph, and that there is an interesting correspondence between the cuts of a graph and the cycles of its dual. For an extreme example, this correspondence is the main reason why the problem MaxCut, which asks for an edge cut of maximum weight and is NP-hard for general graphs, is polynomial-time solvable on planar graphs, even in the more general case where weights may be both positive and negative. Moreover, many standard problems that are used as subroutines in other algorithms admit particularly efficient algorithms that are custom-tailored for planar graphs. For example, matching and flow algorithms fall into this category [MS06, BK09, Eri10]. Finally, planar graphs have often inspired the development of efficient algorithms for larger classes of graphs; for example, for graphs with bounded genus, or for the so-called H-minor free graphs.

The usefulness of all these graph properties strongly varies with the considered problems, and in particular with the role that planarity plays in the problem. If a problem asks to solve a general optimization problem with the additional restriction that the input graph is planar, all the above properties are immediately applicable in order to transform the input graph, reduce it to the same problem on a smaller graph, or to derive a divide & conquer scheme, making many of these kinds of problems solvable more efficiently than

the corresponding general problem. An example of this case is the maximum matching problem, which allows for faster algorithms that are specialized to planar graphs [MS06]. This is, however, not the only role planarity can play in a problem. Because of the useful properties of planar graphs, it may be desirable to require planarity as a side constraint for other optimization problems. This applies to combinatorial optimization problems that ask for a graph with certain properties. A new problem is then obtained by additionally requiring that the solution graph must be planar. Here, the role of planarity is to restrict the set of feasible solutions of an optimization problem, instead of its input. For this kind of problems, it is not at all obvious how any of the nice properties of planar graphs can be applied. Yet another type of problems comes from the area of graph drawing. When drawing a planar graph, it may make a substantial difference for the quality of a drawing which planar embedding of the graph is chosen. Since a planar graph usually has many different embeddings, it is not easy to find an optimal one for a particular drawing style. Here, planarity is a prerequisite for the problem, and the set of feasible solutions consists of a set of planar embeddings of the input graph. We have thus identified three different roles that planarity can have for a problem. This insight is reflected in the structure of this thesis, and it is therefore arranged in two parts. In the first part, we consider combinatorial optimization problems, where either the input is restricted to planar graphs, or where planarity occurs as a side constraint. In the second part, we treat problems of the last type, where the input is a planar graph, and we seek an optimal embedding for a given drawing style.

There exist many problems of these types, and their complexities vary from NP-hard problems over polynomial-time solvable to linear-time solvable problems; we will see several examples throughout this thesis. It therefore seems that planarity has different faces, and it depends on the type of the problem, and in particular the role planarity plays in the problem, which of its many faces becomes apparent when trying to solve the problem.

This work contributes to the theory of planar graphs, and in particular to the classical field of algorithms for planar graphs, by studying several algorithmic questions in which planarity plays different roles. At a higher level, we will also discuss the reasons why some problems are more difficult than others, and relate this to the role that planarity plays in the problem statement. In addition to the basic classification into polynomial-time solvable and NP-hard problems, the main focus of this thesis is to find fast algorithms for the considered problems, in particular linear-time algorithms. Although this varies with the different topics, the main approach is always similar. First, we investigate basic combinatorial properties of the problem under consideration, and, unless it turns out to be NP-hard, we try to give algorithms that are as simple as possible, yet show that the problem is polynomial-time solvable. In a second step, we then refine these algorithms and their implementations until we reach a fast algorithm. In many cases, we achieve optimal linear running time.

Thesis Outline

This thesis consists of two parts. In the first part, we consider problems from the area of combinatorial optimization, of course with a special link to planarity. Here, the two prevalent faces of planarity are that planarity is not a friendly side constraint, and it seems to make problems much more difficult, unless stronger planarity conditions such as planarity of convex geometric graphs are employed. One the other hand, we will experience

a typical example of a problem where the input is restricted to be planar, which often is much better to handle than planarity as a side constraint. The second part focuses on drawing planar graphs and related problems. When drawing planar graphs, the choice of the planar embedding plays a crucial role for the quality of the resulting drawing. We present algorithms that compute particularly good embeddings for different drawing styles. The problems considered in the second part are typical problems of the third type, where the input is a planar graph, and we seek an embedding of this planar graph that is optimal with respect to some quality measure.

Part I: Combinatorial Optimization on Planar Graphs

The first part of the thesis is mainly concerned with combinatorial optimization problems on planar graphs. In particular, we cover three distinct problems.

Graph Augmention (Chapter 3)

One often wants to enhance a given network by adding edges in order to ensure that the final network has a certain property. One such property, which is particularly important for the robustness of a network, is high connectivity. In this thesis, we additionally require that the input graph is planar and stays planar after edges have been added. These constraints make the problem more difficult than its counterpart without the planarity constraint.

We study several variants of this problem with different types of planarity constraints and different variants of connectivity criteria.

Switch Graphs (Chapter 4)

Switch graphs are a generalization of ordinary graphs. A switch consists of a set of edges that share a common vertex. A configuration picks exactly one edge from each switch. Hence, a switch graph describes a family of graphs, and a configuration describes a concrete member of such a family. Switches are common building blocks in telephone networks and IT infrastructure networks. Given a switch graph, one is interested in finding out whether the family of graphs it describes contains a graph with a certain property. For example, one would like to know whether it is possible to find a switch configuration so that packages between two given computers can be routed directly, without changing the configuration of any switches in between.

We study several problems of this type. In particular, we consider the variants where the desired properties are Eulerian, bipartite, planar and connected.

Large Matchings in Planar Graphs (Chapter 5)

A matching is a subset of edges of a graph such that each node is incident to at most one of the edges in the set. Finding large matchings, and in particular matchings of maximum cardinality, is a classical problem in combinatorial optimization. Nishizeki and Baybars [NB79] studied matchings in graphs with fixed minimum degree and proved lower bounds on the size of maximum matchings in such graphs. It has, however, been unclear whether a matching whose size matches the lower bound, which is known to exist, can be computed more efficiently than a maximum matching.

We show how fixed minimum degree in planar graphs can be exploited to find matchings with a guaranteed minimum size, quickly. In particular, for graphs with minimum degree 3, we achieve the tight bound of Nishizeki and Baybars.

Part II: Embeddings of Planar Graphs

The second part of this thesis is concerned with the problem of finding embeddings of planar graphs that are especially well-suited for certain visualization styles. These types of problems are inherently difficult, as planar graphs generally have an exponential number of planar embeddings. We consider embedding problems for different drawing styles, in particular for topological drawings, where edges are drawn as arbitrary curves, and for orthogonal drawings, where edges must be composed of axis-parallel line segments. It will turn out that the planar embeddings of a given graph have a lot of structure that can often be exploited to obtain surprisingly simple solutions.

Planarity of Partially Embedded Graphs (Chapters 6 and 7)

In a topological drawing of a graph, nodes are represented by points and edges are represented by Jordan curves between their endpoints. By definition, a graph is planar if it admits a planar drawing in this way. We study an extension problem for planar drawings, where a subgraph is already drawn and the question is whether the remainder of the graph can be drawn in a planar way without modifying the drawing of the given subgraph. For straight-line drawings this problem is known to be NP-hard [Pat06]. For topological drawings, this drawing problem is actually equivalent to an embedding problem, where the given drawing of a subgraph describes an embedding of the subgraph. This leads to the notion of partially embedded graphs.

We show that this problem can be solved in polynomial, even linear time; see Chapter 6. This is particularly remarkable, as it is a typical partial-solution extension problem, which are often much more difficult than their counterparts without a given partial solution.

Following the example of Kuratowski's famous characterization of planar graphs, we introduce a containment relation for partially embedded graphs analogous to the usual minor containment and characterize the planar partially embedded graphs via forbidden substructures; see Chapter 7.

Together, these two chapters result in an efficient certifying algorithm for the partial embedding problem, which in the positive case outputs a corresponding embedding, and in the negative case extracts a forbidden substructure, as a proof for non-embeddability.

Simultaneous Embeddings (Chapter 8)

Given two (or more) graphs on the same vertex set, one is often interested in comparing these graphs. One approach to compare them effectively is to find a drawing of the two graphs that highlights similarities as much as possible. Since the union of two planar graphs is generally not planar, this chapter steps slightly beyond the usual notion of planarity. We study the problem of finding simultaneous embeddings with fixed edges, where the vertices of both graphs are drawn at the same positions, common edges are represented by the same curves, and the induced drawing of each single graph is planar. Hence, in a simultaneous embedding with fixed edges, only edges belonging to different graphs may cross.

Although a lot of research effort went into this problems over the last few years, its complexity is still open. We study the complexity of this problem for special cases of the input. In particular, we derive a linear-time algorithm for the case when the intersection of the two graphs is biconnected. Before, such a result was only known for more restricted cases, when one of the graphs contains at most one cycle, or when both graphs are outerplanar.

Orthogonal Drawings with Flexibility Constraints (Chapter 9)

In orthogonal drawings all edges are composed of axis-parallel straight-line segments, which automatically yields good angular resolution. Since edges with many bends degrade the readability of a drawing, a typical optimization criterion is the total number of bends or the maximum number of bends per edge. On the one hand, it is known that deciding whether a drawing without bends exists is NP-hard. On the other hand, two bends per edge always suffice. The complexity of deciding whether a given graph can be drawn with at most one bend per edge was unknown.

We introduce the notion of flexibility, which allows us to specify, for each edge individually, its maximum number of bends. Obviously, this contains the problem of drawing a graph with at most one bend per edge, but it is more general. The possibility to allow fewer bends for important edges and to give additional freedom to less important ones, potentially yields clearer drawings. We show that the problem of deciding whether a graph admits a drawing adhering to given flexibility constraints can be decided efficiently if every edge may have at least one bend. By carefully recycling partial solutions we achieve quadratic running time for this problem.

Chapter 2

Preliminaries

This chapter introduces basic concepts and notations that are used throughout this thesis. We do not intend to give a complete introduction to the topic, rather we briefly introduce these notions, and then focus on the most important interconnections in the context of this thesis. Therefore we assume some familiarity with basic mathematical concepts, such as naive set theory and the Big-O-notation. Good introductions to graphs and graph algorithms are provided in the book by Diestel [Die10] and in "Combinatorial Optimization" by Korte and Vygen [KV08]. "Introduction to Algorithms" [CLRS01] is a good reference for the Big-O-notation and contains many basic algorithms. The book "Computers and Intractability" by Garey and Johnson [GJ79] provides a nice introduction to the theory of NP-completeness, as well as a rich compendium of NP-hard problems.

2.1. Graphs and Related Concepts

A *directed graph* is a tuple $G = (V, E)$ consisting of a set V of *vertices* (or *nodes*) and an *edge set* E that forms a binary relation on V, that is, $E \subseteq V \times V$. The elements of E therefore are ordered pairs (u, v) of vertices in V. We will denote the vertex set and the edge set of a graph by $V(G)$ and $E(G)$, respectively. For brevity, we also use the notation uv to denote the edge (u, v). For an edge $e = uv$ we say that e *connects* u and v, such that u is the *source* and v is the *target* of e and that u and v are the *endpoints* of e. An *undirected graph* is essentially the same, except that the relation defined by E is symmetric and therefore consists of unordered pairs $\{u, v\} \subseteq V$, and hence $uv = vu$ for undirected graphs. A *graph* is either a directed graph or an undirected graph. We will usually denote the cardinality of the vertex set V by $n := |V|$, and the cardinality of the edge set E by $m := |E|$.

A vertex u is *incident* to an edge e if u is an endpoint of e. Two distinct vertices u and v are *adjacent* if there is an edge e that is incident to both u and v, that is, if there is an edge e that connects u and v. Note that our definition allows for edges whose endpoints coincide. Such an edge $e = vv$ that is incident to the same vertex twice is called a *loop*. A graph that does not contain loops is *loop-free*.

Sometimes we also study *multi-graphs*, where E is a multi-set and may contain an element multiple times. For an undirected graph, two edges e and e' with $e \neq e'$ that have the same endpoints are called *parallel*. For directed graphs the edges e and e' need to have the same source and target to be parallel. A graph is called *simple* if it has neither loops nor parallel edges. Graphs that may contain parallel edges or loops are also called

non-simple graphs and we will state their usage explicitly in this thesis. Unless stated otherwise, all graphs in this thesis are assumed to be simple undirected graphs.

For a vertex v denote by $E(v)$ the set of edges that are incident to v. The number of edge incidences a vertex v has, is called its *degree* and is denoted by $\deg(v)$. Note that for loop-free graphs, it holds that $\deg(v) = |E(v)|$, while for graphs with loops we have to count the loops twice. For directed graphs we further distinguish the *out-degree* $\deg^{\rightarrow}(v)$ and the *in-degree* $\deg^{\leftarrow}(v)$, denoting the number of edges for which v is a source and a target, respectively. For every vertex v we have $\deg(v) = \deg^{\leftarrow}(v) + \deg^{\rightarrow}(v)$. If necessary, we will provide the graph to which these notations refer as an index; for example $\deg_G(v)$. The *neighbors* of u are all the vertices that are adjacent to u, and are denoted by $N(u) = \{v \mid uv \in E \text{ or } vu \in E\}$. A vertex with degree 0 is *isolated*.

Paths, Cycles and Trees. A *walk* in a graph $G = (V, E)$ is a sequence W of vertices and edges such that $W = v_0, e_1, v_1, e_2, \ldots, e_k, v_k$ with $e_i = v_{i-1}v_i$ for $i = 1, \ldots, k$, that is any two consecutive vertices of W are connected by an edge. A walk *contains* a vertex v if $v = v_i$ for some $i = 0, \ldots, k$ and it *contains* an edge e if $e = e_i$ for some $i = 1, \ldots, k$. The vertices v_0 and v_k are *endvertices* of W and the nodes $v_1, \ldots, v_{k-1}$ are *inner nodes* of W. We say that W *starts* at v_0 and *ends* at v_k and W *connects* v_0 to v_k.

A *path* is a walk that contains each edge at most once and a path that also contains each vertex at most once is *simple*. A *subpath* of a path P is a path that is contained in P. For simple paths a subpath can be described by its first and last vertex. A path whose endpoints coincide is a *cycle*, and it is a *simple cycle* if each vertex is contained at most once, except for the endpoint v_0, which occurs twice. A graph that does not contain any cycles is *acyclic*. Note that for simple graphs we can omit the edges from the sequence and describe paths as sequences of vertices only. A graph is *connected* if there exists a path between any two vertices.

A directed acyclic graph is also called a *DAG*. An undirected acyclic graph is a *forest*. A connected forest is a *tree*. Every pair of vertices in a tree T is connected by a unique path in T. Vertices with degree 1 are called *leaves*, and every tree has at least two leaves. Often it is convenient to root a tree, that is to pick a certain node as the root in the tree. This gives rise to a parent relationship for all other nodes. Namely, if r is the root and $v \neq r$ is another vertex of the tree, then the first vertex after v on the (unique) path from v to r is the *parent* of v. Once a root is chosen, every vertex except the root has a unique parent in this way. All other neighbors of a vertex are *children*, and hence each vertex is the parent of all its children.

Subgraphs and Elementary Graph Operations. A *subgraph* of a graph $G = (V, E)$ is a graph G' with $V(G') \subseteq V$ and $E(G') \subseteq E$. For a subset $V' \subseteq V$ of vertices the *induced* subgraph of V' in G is the graph $(V', \{uv \in E(G) \mid u, v \in V'\})$, and it is denoted by $G[V']$. A subgraph G' of G is *spanning* if $V(G) = V(G')$. In particular a *spanning tree* of G is a tree $T = (V, E')$ with $E' \subseteq E$.

Often, it will be necessary to modify a graph. We briefly introduce some basic operations for modifying graphs. Let $G = (V, E)$ be a graph and let $E' \subseteq E$ be any subset of edges. We say that the graph $G' = (V, E \setminus E')$ is obtained from E by *removing* the edges in E' and we denote this graph by $G - E'$. For a set $V' \subseteq V$ we say that the graph $G[V \setminus V']$ is obtained from G by *removing* the vertices in V' and denote this graph by $G - V'$. Note that this is the graph that is obtained from G by removing all edges that are incident to vertices of V' and then removing the (now isolated) vertices of V'. We simply write $G - e$ for $G - \{e\}$ and $G - v$ for $G - \{v\}$. Similarly, it is sometimes useful to add edges to a

graph $G = (V, E)$ and we write $G + uv$ for the graph $(V, E \cup \{uv\})$, where u and v are two vertices of V.

Another useful operation is that of *subdividing an edge* $e = uv$ of a graph $G = (V, E)$. This operation inserts a new vertex w and replaces the edge uv by two edges uw and wv. A *subdivision* of a graph G is a graph G' that can be obtained from G by a sequence of edge subdivisions. Given a graph G together with edge $e = uv$ the graph G/e is the graph that is obtained from G by identifying the endpoints of e. This operation is called *contracting* edge e. A *minor* of a graph G is any graph that can be obtained from a subgraph of G by successively contracting edges. If H is a minor of G, we also say that G *contains* H as a minor.

Connectivity and the Block–Cutvertex Tree. Recall that a graph is connected, if there exists a path between any pair of vertices. The maximal connected subgraphs of a graph are called its *connected components*. An edge whose removal increases the number of connected components is called a *bridge*.

A graph G is *k-vertex connected* (*k-edge connected*) if removing at most k vertices (edges) leaves G connected. We will use the term *k-connected* as an abbreviation for k-vertex connected. A graph is 2-edge connected, if it does not contain any bridges, therefore we sometimes also use the term *bridge-connected* for such graphs. A set of vertices whose removal disconnects a graph is called a *separator*. Separators of size 1, 2, and 3 are called *cutvertices*, *separation pairs* and *separating triplets*, respectively. A connected graph is 2-connected if it does not have a cutvertex and it is 3-connected if it does not have a separation pair. Graphs that are 2-connected or 3-connected are also called *biconnected* and *triconnected*, respectively.

The maximal biconnected subgraphs of a graph G are called *blocks*. Each edge and each vertex that is not a cutvertex belongs to exactly one block, while cut vertices are shared between several blocks. Note that in particular, each bridge forms a block of its own. To capture the relationship of the blocks of a graph, we use the *block–cutvertex tree*, which contains as nodes the blocks and cutvertices of G. A cutvertex v and a block B are connected by an edge if v belongs to B. We use the term *nodes* for vertices of the block–cutvertex tree in order to distinguish them from the vertices of G.

2.2. Drawings and Planarity

A *(topological) drawing* of a graph $G = (V, E)$ is a mapping Γ that maps the vertices of G to distinct points in the plane and maps edges to simple Jordan curves connecting their endpoints. A drawing is *planar* if the curves representing the edges do not cross but, possibly, at common endpoints. A graph is *planar* if it admits a planar drawing. A planar drawing Γ determines a subdivision of the plane into connected regions, called *faces*, and a circular ordering of the edges incident to each vertex, called *rotation scheme*. Moreover, one of the faces is unbounded and is called the *external face* or the *outer face*. The remaining faces are called *internal faces*.

Visiting the (not necessarily) connected border of a face f of Γ in such a way that f is to the left, we determine a set of circular lists of vertices, the *face boundary* of f. Two drawings are equivalent if they have the same rotation schemes, the same face boundaries, and the same external face. A *planar embedding* is an equivalence class of planar drawings. For connected graphs an embedding is completely described by the rotation scheme and

the external face. For biconnected graphs, each face is bounded by a simple cycle, that is each face boundary consists of a single circular vertex list without duplicates.

Embedding Graphs on the Sphere. In this thesis we will sometimes also consider drawings on the sphere. Although it is in principle also possible to draw graphs on arbitrary surfaces, this is beyond the scope of this thesis. Drawing graphs on the sphere is closely related to graph drawing in the plane. In fact, a graph admits a planar drawing in the plane if and only if it admits a planar drawing on the sphere. The *stereographic projection* provides a transformation of such drawings, mapping drawings on the sphere into the plane. Its inverse can be used to project a drawing in the plane onto the sphere. As the sphere is compact, all its faces are bounded and the distinction between the internal faces and the external face vanishes. When projecting back to the plane, the face containing the north-pole becomes the external face. Since, by simple rotation of the sphere, any face can be made to contain the north-pole, this shows that the external face can be chosen arbitrarily without altering the rotation scheme and the face boundaries.

Other drawing styles. So far we have only considered topological drawings, where edges of a graph are represented by arbitrary Jordan curves. This is, however, not the only choice, and we can further restrict the curves that may be used in a drawing. Two widespread alternatives are *straight-line drawings* and *orthogonal drawings*. In a straight-line drawing, the curves representing the edges are the straight-line segments between their endpoints. For orthogonal drawings, each curve consists of a piecewise axis-parallel curve. A graph together with a straight-line drawing is also called a *geometric graph*. Naturally, these drawing conventions give rise to corresponding planarity definitions, by defining a graph to be planar if it admits a corresponding planar drawing. Fortunately, these planarity definitions are not vastly different. By a classical result of Kőnig, Fáry and Stein, any planar graph admits a planar straight-line drawing, and hence the topological and the geometric planarity definitions coincide. For orthogonal drawings it is clear that any vertex may have degree at most 4. Tamassia [Tam87] showed that any planar graph with maximum degree 4 admits an orthogonal drawing. Hence, the orthogonal planarity definition is the usual planarity definition restricted to graphs with maximum degree 4. In both cases even a given combinatorial embedding can be preserved.

By further restricting the drawing conventions we can obtain other notions of planarity, for example, by additionally requiring that the vertices are in convex position. For topological drawings, again, this notion of planarity coincides with the standard planarity definition [PW01]. However, for geometric graphs this makes a difference. A *convex geometric graph* is a geometric graph whose vertices are in convex position. The set of graphs that admits a planar straight-line drawing with the vertices in convex position is exactly the class of *outerplanar graphs*, where a graph is *outerplanar* if it admits an embedding such that all vertices are on the external face.

Basic Properties of Planar Graphs. Planarity is a very strong property for graphs and the main topic of this thesis. Here we highlight a few connections of planarity to connectivity and other graph properties. A graph is planar if and only if all its blocks are planar. We will see later in Section 2.3 that a graph is planar if and only if its triconnected components are planar. Note that given any planar embedding $\mathcal{E}$ of a graph G another planar embedding of G can be obtained by simultaneously reversing the circular lists of incident edges of all vertices. This embedding is called the *flip* of $\mathcal{E}$. By a classical theorem of Whitney [Whi32], the embedding of a 3-connected planar graph is fixed up to flip and the choice of the external face. Therefore also any subdivision of a 3-connected graph has

a (in this sense) unique embedding.

Although the structure of the faces, that is, the set of face boundaries, grossly changes when varying the embeddings, the number of faces does not. For connected graphs it only depends on the number of vertices and edges of the graph. More precisely, for a planar graph with n vertices, m edges and f faces we have that $n + m - f = 2$, which is known as Euler's formula. This formula can be used to show that a simple planar graph with n vertices has at most $3n - 6$ edges (and therefore at most $2n - 4$ faces). In particular, the number of edges a simple planar graph can have is linear in the number of vertices. Note that a graph is planar if and only if the graph obtained by removing parallel edges is planar. Further, this shows that the average degree of a simple planar graph is less than 6 and therefore any simple planar graph contains a vertex of degree at most 5.

Planarity is a property that is preserved by several operations. For instance, any subgraph of a planar graph is again planar, and by restricting the rotation scheme of a planar embedding $\mathcal{E}$ of a graph G to the edges in a subgraph $H \subseteq G$, we obtain a planar embedding $\mathcal{E}|_H$ of H induced by $\mathcal{E}$. Similarly, any subdivision G^+ of G is planar and $\mathcal{E}$ also induces a planar embedding on G^+. Contracting an edge $e = uv$ can be seen as a continuous motion of the endpoints of the curve representing e in a drawing. Therefore, contracting an edge $e = uv$ in a graph G with a planar embedding $\mathcal{E}$ yields not just the graph G/e, but along with it an induced planar embedding $\mathcal{E}/e$ with the following properties. For each vertex of G/e distinct from u and v, the circular ordering of the incident edges is the same as in $\mathcal{E}$. For the vertex w resulting from the contraction of u and v, the edges that were originally incident to u and v keep their circular order and both form intervals in the circular ordering around w. Moreover, the interval stemming from the edges incident to v occurs in the circular ordering around u at the former position of the edge uv. Symmetrically, the edges formerly incident to u are inserted at the former position of uv in the ordering around v. In particular, every minor of a planar graph is again planar. Kuratowski [Kur30] showed that a graph is planar if and only if it contains neither K_5, nor $K_{3,3}$ as a minor. This completely characterizes the class of planar graphs by *forbidden minors*. The first linear-time algorithm for checking whether a graph is planar is due to Hopcroft and Tarjan [HT74]. Today, there are several different linear-time planarity tests available [BM04, dFdMR06, HT08]. In the case that the input graph is planar, these algorithms construct a planar embedding. Otherwise, they extract a subdivision of K_5 or $K_{3,3}$, which serves as a certificate for non-planarity.

Finally, for a planar graph with a fixed planar embedding there exists a notion of a *dual graph*. Let $G = (V, E)$ be a planar graph with a fixed planar embedding, and let F denote the set of faces of G in the given embedding. The dual $G^\star$ of G is a graph whose vertices are the faces of G, and whose edges are dual to those in G in the following sense. Any edge e of G is incident to two faces f_1 and f_2 in the embedding of G. We denote by $e^\star = f_1 f_2$ the edge between these two faces. The dual graph then is the graph $G^\star = (F, E^\star)$, where $E^\star = \{e^\star \mid e \in E\}$ is the set of dual edges of E.

2.3. The SPQR-tree

Similar to the decomposition of a (connected) graph into biconnected components, it is possible to decompose a biconnected graph into its triconnected components [HT73]. Again the relationships among the triconnected components can be modeled as a tree, called the *SPQR-tree*. In particular, the SPQR-tree allows to concisely represent the

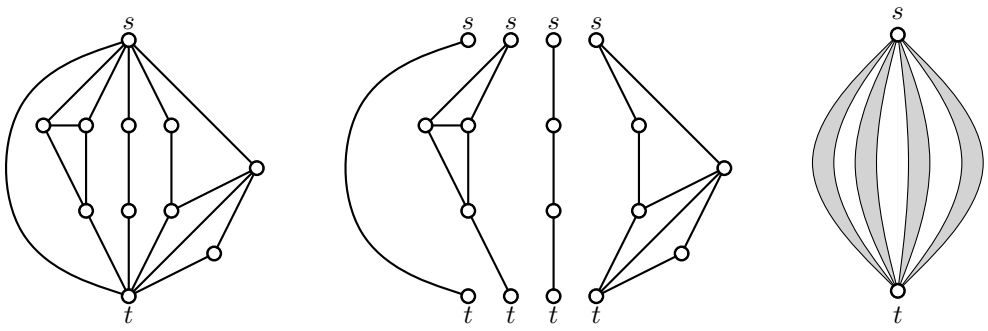


Figure 2.1.: Example of a split pair $\{s, t\}$ (a) and its maximal split components (b). The skeleton of a P-node showing the arrangement of the maximal split components for the split pair $\{s, t\}$ (c); the virtual edges are drawn as thick grey edges. The graphs in (b) are the expansion graphs of the virtual edges in the skeleton.

possible planar embeddings of a given biconnected planar graph. The SPQR-tree was introduced by Di Battista and Tamassia [DT96b, DT96a], based on ideas by Bienstock and Monma [BM89, BM90]. We first give a description of the properties of the SPQR-tree and describe its construction. Afterwards, we show how it can be used to represent all planar embeddings of a biconnected planar graph.

A graph G with vertices s and t is *st-biconnectible* if adding the edge st yields a biconnected graph. A *split pair* of G is either a separation pair of G or a pair of adjacent vertices. A *maximal split component* of a split pair $\{u, v\}$ is either an edge uv or a maximal subgraph C of G such that C contains u and v and $\{u, v\}$ is not a split pair of C. Note that every vertex $w \neq u, v$ belongs to exactly one maximal split component. We call the union of any number of maximal split components of $\{u, v\}$ a *split component* of $\{u, v\}$; note that indeed a split component may consist of several maximal split components. This sounds strange at first, and the reason is that "maximal" rather refers to the splitting, not to the components. We decided to keep the name for consistency with the literature. See Figure 2.1 for an example of a split pair and the corresponding split components.

The SPQR-tree of a biconnected graph G is a tree $\mathcal{T}$ whose leaves correspond bijectively to the edges of G. It is constructed by recursively decomposing G along split pairs. We refer to the vertices of $\mathcal{T}$ as *nodes* in order to distinguish them from the vertices of G. Each node μ is associated with a multigraph $\text{skel}(\mu)$, the *skeleton* of μ, which can be seen as a sketch of the graph from a certain viewpoint. See for example the skeleton in Figure 2.1c, which sketches the graph from Figure 2.1a from the viewpoint of the split pair $\{s, t\}$. The construction is such that each skeleton forms either a cycle, a set of parallel edges on two vertices, or a triconnected graph. Each vertex of $\text{skel}(\mu)$ is also a vertex of G, and each edge uv in $\text{skel}(\mu)$ represents a corresponding split pair $\{u, v\}$ in G. The edges of the skeletons are either *virtual edges* representing a subgraph of G containing the endpoints of the virtual edge or *real edges*, which are edges that also belong to G.

The leaves of $\mathcal{T}$ are also called Q-nodes. The skeleton of the Q-node corresponding to an edge uv contains the two vertices u and v and two parallel edges between u and v, one real edge representing the edge uv and one virtual edge representing the rest of the graph. Note that in our definition of the SPQR-tree only the skeletons of Q-nodes contain real edges, all other edges of skeletons are virtual edges. An SPQR-tree $\mathcal{T}$ has three types of internal nodes, namely S-nodes, P-nodes, and R-nodes, depending on the type of the skeleton of

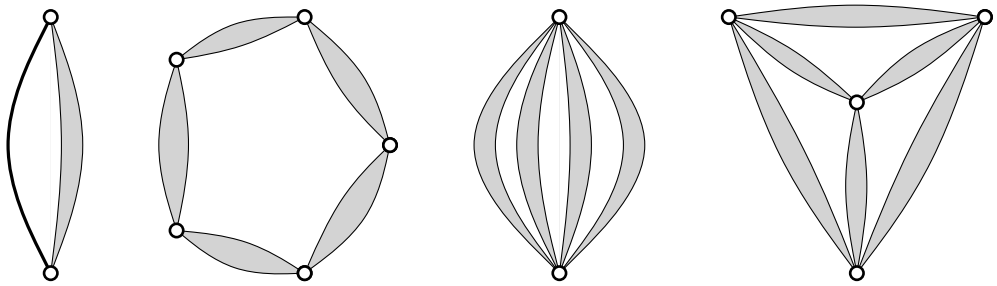


Figure 2.2.: Skeletons of different node types, virtual edges are thick and grey, real edges are fat black edges. Skeletons of a Q-node (a), an S-node (b), a P-node (c), and an R-node (d).

the node. An *S-node* (or *series node*) is a node whose skeleton is a cycle of length $k \geq 3$. A *P-node* (or *parallel node*) is a node whose skeleton has two vertices and $k \geq 3$ parallel edges. An R-node (or *rigid node*) is a node whose skeleton is a simple 3-connected graph. It is assumed that no two S-nodes and no two P-nodes are adjacent in $\mathcal{T}$. Figure 2.2 shows examples for the skeletons of different types of nodes.

Two distinct skeletons $\text{skel}(\mu)$ and $\text{skel}(\mu')$ share vertices if and only if μ and μ' are adjacent in $\mathcal{T}$. In this case they share exactly two vertices u and v and both contain the virtual edge uv. Now suppose that $\mathcal{T}_1, \ldots, \mathcal{T}_k$ are the subtrees of $\mathcal{T}$ adjacent to the node μ, that is, they are the connected components of $\mathcal{T} - \mu$. Each of these subtrees is associated bijectively with a virtual edge e_i of $\text{skel}(\mu)$. Moreover, each of these subtrees defines a graph G_i formed by the edges of G represented by the Q-nodes of $\mathcal{T}_i$ together with the vertices of G incident to these edges. Let $e_i = uv$. The graph G_i is connected, contains the two vertices u and v, and does not contain any other vertex of $\text{skel}(\mu)$. If the graphs G_i and G_j with $i \neq j$ share a vertex v, then v must be a common endpoint of the edges e_i and e_j in $\text{skel}(\mu)$. Therefore, the graphs $G_1, \ldots, G_k$ are an edge-disjoint decomposition of G, and $\text{skel}(\mu)$ is obtained from G by contracting each G_i into a single edge e_i. We say that the graph G_i *projects* into e_i and that G_i is the *expansion graph* of e_i. For an edge or a vertex of G_i we also say that it *belongs to* e_i.

The SPQR-tree always exists and is unique. Before we describe its construction and its relation to planarity, we need some further definitions.

Construction of the SPQR-tree. The rooted SPQR-tree is defined with respect to a *reference edge* of G. It describes a recursive decomposition of G along its split pairs; the endpoints of each virtual edge e of a skeleton form a split pair splitting off the expansion graph of e. Note that rooting the tree at a node determines for each node μ except for the root one special virtual edge, namely the one that $\text{skel}(\mu)$ shares with its parent. Usually, SPQR-trees are rooted at Q-nodes, and the construction is such that the tree is rooted at the Q-node corresponding to the reference edge.

To obtain the *pertinent graph* of μ, denoted by $\text{pert}(\mu)$, we replace each virtual edge e_i of $\text{skel}(\mu)$, except for the one that μ shares with its parent, by its expansion graph. The remaining virtual edge then represents the rest of the graph in $\text{pert}(\mu)$. Note that the pertinent graph of the root is G itself.

We are now ready to describe the construction of the SPQR-tree. Given a biconnected graph G with reference edge $e = u'v'$, the SPQR-tree is recursively defined as follows. At each step, a split component $G^\star$, a pair of vertices $\{u, v\}$, and a node ν in $\mathcal{T}$ are given.

A node μ corresponding to $G^\star$ is introduced in $\mathcal{T}$ and attached to its parent ν. The vertices u and v are the *poles* of μ and they are denoted by $u(\mu)$ and $v(\mu)$, respectively. The decomposition possibly recurs on some split components of $G^\star$. Initially, we set $G^\star = G - e$, $\{u, v\} = \{u', v'\}$ and ν is a Q-node, corresponding to e. Figure 2.3 shows a complete example illustrating the construction of an SPQR-tree.

Base Case: If $G^\star$ consists of a single edge from u to v, then $\mathcal{T}$ is a single Q-node whose skeleton is $G^\star$ itself.

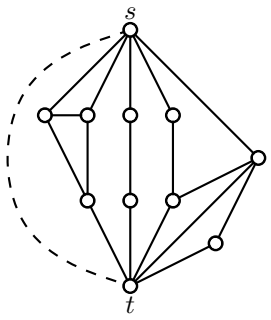
Series Case: If $G^\star$ is not biconnected, let $v_1, \ldots, v_{k-1}$, $k \geq 2$, be its cutvertices and let $G_1, \ldots, G_k$ be its blocks in the order from u to v. Then μ is an S-node and the graph skel(μ) is the path $e_1, \ldots, e_k$, where the virtual edge e_i connects v_{i-1} with v_i ($i = 2, \ldots, k-1$), e_1 connects u with v_1, and e_k connects v_{k-1} with v. The decomposition recurs on the split components corresponding to each of the virtual edges $e_1, e_2, \ldots, e_{k-1}, e_k$ with μ as parent node, and with $\{u, c_1\}, \{c_1, c_2\}, \ldots, \{c_{k-2}, c_{k-1}\}$, $\{c_{k-1}, v\}$ as pair of vertices, respectively.

Parallel Case: If $\{u, v\}$ is a split pair of $G^\star$ with maximal split components $G_1, \ldots, G_k$, $k \geq 2$, then μ is a P-node. The graph skel(μ) consists of k parallel virtual edges $e_1, \ldots, e_k$ between u and v. The decomposition recurs on $G_1, \ldots, G_k$ with $\{u, v\}$ as pair of vertices for every graph, and with μ as parent node.

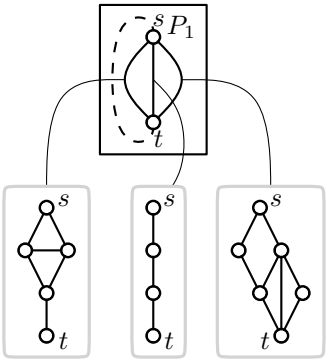
Rigid Case: If none of the above cases applies, the goal of the decomposition step is to partition $G^\star$ into the smallest number of split components and recurring on each of them. We need some further definitions. Given a maximal split component G' of a split pair $\{s, t\}$ of $G^\star$, a vertex $w \in V(G')$ *properly belongs to* G' if $w \neq s, t$. Given a split pair $\{s, t\}$ of $G^\star$, a maximal split component G' of $\{s, t\}$ is *internal* if neither u nor v (the poles of $G^\star$) properly belong to G'. Otherwise, it is *external*. A *maximal split pair* $\{s, t\}$ of $G^\star$ is a split pair of $G^\star$ that is not contained in an internal maximal split component of any other split pair $\{s', t'\}$ of $G^\star$. Let $\{u_1, v_1\}, \ldots, \{u_k, v_k\}$ be the maximal split pairs of $G^\star$, $k \geq 2$ and, for $i = 1, \ldots, k$ let G_i be the union of all the internal maximal split components of $\{u_i, v_i\}$. Observe that each vertex of $G^\star$ either properly belongs to exactly one G_i or belongs to some maximal split pair $\{u_i, v_i\}$. The node μ is an R-node. The graph skel(μ) is the graph obtained from $G^\star$ by replacing each subgraph G_i with the virtual edge $e_i = u_i v_i$. The decomposition recurs on each G_i with μ as parent node and with $\{u_i, v_i\}$ as pair of vertices.

For each node μ of $\mathcal{T}$ with poles u and v, the construction of skel(μ) is completed by adding a virtual edge uv representing the rest of the graph, that is the graph obtained from G by removing all the vertices of pert(μ) except for its poles, together with their incident edges. Note that his makes all skeletons biconnected, and in particular the skeletons of R-nodes become triconnected. Moreover, by construction of the SPQR-tree, the expansion graph of a virtual edge uv of an S-node μ does not contain a cutvertex separating u and v, and hence no two S-nodes are adjacent. Similarly, the expansion graph G_{uv} of a virtual edge uv of a P-node is either a single edge or it does not contain the edge uv and $G_{uv} - \{u, v\}$ is connected, that is there are no two adjacent P-nodes in $\mathcal{T}$.

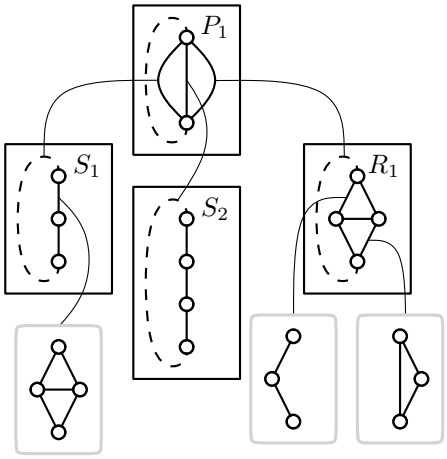
Although at first it seems that the construction might depend on the choice of the reference edge, this is not the case. Instead, changing the reference edge from an edge e to a different edge e' corresponds to rerooting the tree at the Q-node corresponding to e'. It is important to note that the SPQR-tree itself, the skeletons and the expansion graph of an



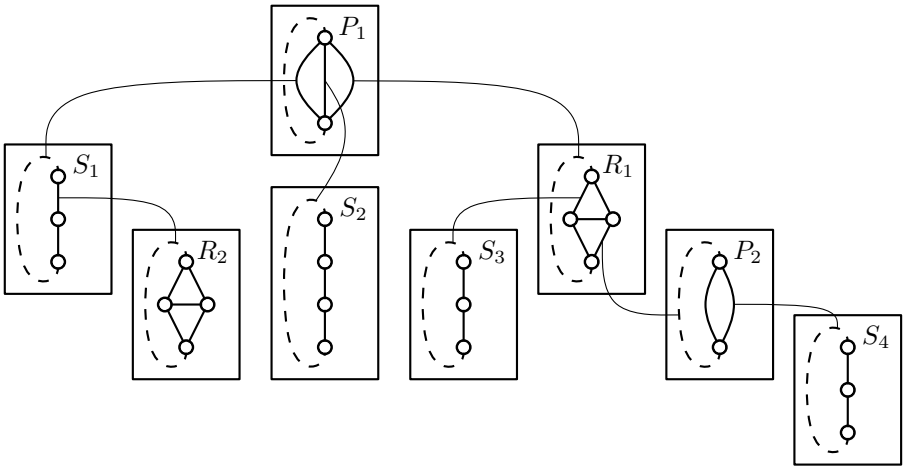
(a) The original graph, the reference edge is drawn dashed.



(b) Decomposition along the split pair $\{s, t\}$, resulting in a P-node and its expansion graphs.



(c) Further decomposition of the remaining expansion graphs.



(d) The final SPQR-tree.

Figure 2.3.: Example of the construction of an SPQR-tree, for clarity, all Q-nodes are omitted. The reference edge of the original graph is dashed and for each skeleton $\mathfrak{S}$, the edge that $\mathfrak{S}$ shares with its parent is dashed and the two edges are connected by a curve. Graphs enclosed in thick grey boxes with rounded corners are expansion graphs and are further decomposed in the next step.

edge only depend on the structure of the original graph G, while the pertinent graph and the poles of a skeleton also depend on the choice of the reference edge. As we will see later, rooting the tree at the Q-node representing e amounts to considering only embeddings of the graph for which e lies on the external face. The unrooted version corresponds to neglecting the information about the external face and therefore models embeddings of graphs on the sphere.

The SPQR-tree and Planarity. Until now, all the definitions of the SPQR-tree apply to general biconnected graphs, not just to biconnected planar graphs. As mentioned above, each skeleton $\text{skel}(\mu)$ can be obtained as a minor by contracting the expansion graphs of its edges. Hence, for a planar graph all skeletons are planar and in particular a planar embedding $\mathcal{E}$ of a graph G induces in this way a planar embedding $\mathcal{E}(\mu)$ for each skeleton $\text{skel}(\mu)$. Note that if the SPQR-tree is rooted at an edge that is incident to the outer face, then for each node μ of $\mathcal{T}$ (except the root), the edge that $\text{skel}(\mu)$ shares with its parent lies on the external face of $\mathcal{E}(\mu)$. Conversely, if a planar embedding $\mathcal{E}_\mu$ for each skeleton $\text{skel}(\mu)$ is fixed, a planar embedding $\mathcal{E}$ of G can be constructed with the property that $\mathcal{E}(\mu)$ coincides with $\mathcal{E}_\mu$ for each node μ of $\mathcal{T}$.

Two skeletons of adjacent nodes μ and μ' can be merged as follows. Since μ and μ' are adjacent, their skeletons share two vertices u and v and the virtual edge uv. To merge $\text{skel}(\mu)$ and $\text{skel}(\mu')$ we first identify their common virtual edge uv and then remove it from the graph. The embeddings $\mathcal{E}_\mu$ and $\mathcal{E}_{\mu'}$ together induce a unique planar embedding of the merged graph. Exhaustively merging all skeletons recreates the original graph G, and hence, along with, it the desired embedding $\mathcal{E}$.

This shows that a biconnected graph G is planar if and only if all skeletons of its SPQR-tree are planar. Moreover, if G is a planar graph and $\mathcal{T}$ is its SPQR-tree, any planar embedding of G induces a unique planar embedding of each skeleton of $\mathcal{T}$, and conversely, by specifying a planar embedding for each skeleton, we obtain a unique planar embedding of G. Note that the skeletons of S-nodes, P-nodes and Q-nodes are always planar and hence a graph is planar if and only if all its R-node skeletons are planar. Further, considering the skeletons we can also get more information on how many different possible embeddings a planar graph may have. For S- and Q-nodes the skeleton has only a single embedding. The skeletons of R-nodes are 3-connected planar graphs and therefore have a unique embedding up to a flip. Finally, a P-node whose skeleton consists of k parallel edges admits $(k-1)!$ distinct circular orderings of its edges. This also allows us to compute the number of embeddings of a planar graph. If the SPQR-tree of a biconnected planar graph G contains r R-nodes and p P-nodes, such that the skeleton of the i-th P-node consists of p_i parallel edges, then G has $2^r \cdot \prod_{i=1}^{p}(p_i - 1)!$ distinct planar embeddings, up to the choice of the external face. Since any of the involved numbers may be of linear size, such a graph usually has an exponential number of different planar embeddings.

Finally, the SPQR-tree of a biconnected planar graph can be computed in linear time [GM00], making it a useful tool for fast algorithms.

2.4. Complexity

While the goal of this thesis is to provide efficient algorithms for the considered graph-theoretic problems, for some problems we did not succeed in constructing such algorithms. In many of these cases we instead show that it is unlikely that the problem admits an

efficient algorithm by showing that it is NP-hard. In this section we briefly sketch the basic ideas behind these arguments and why it is believed that these problems do not admit efficient solutions.

The classes $\mathcal{P}$ and $\mathcal{NP}$. The class $\mathcal{P}$ contains all problems that admit an *efficient algorithm*. An algorithm is efficient, if its running time is polynomially bounded by its input size, that is, if the input size is n, its running time is in $O(n^c)$ for some fixed constant c. Note that this does not, in general, mean that such an algorithm is particularly useful: although an algorithm with running time $\Theta(n^{100})$ is efficient, it will most probably be unusable in practice as it would probably not even solve the smallest instances within any reasonable time frame.

The class $\mathcal{NP}$ contains all problems for which a given solution can be verified in polynomial time. Clearly, we have $\mathcal{P} \subseteq \mathcal{NP}$.

To understand the difference between these two concepts it is helpful to consider a simple example. Given a natural number N, do there exist natural numbers $n_1, n_2 > 1$ such that $n_1 \cdot n_2 = N$? While it may be difficult to find out whether such a factoring exists, given a possible solution of two numbers n_1 and n_2 even a school kid can easily check whether they form a solution to the problem, simply by computing their product. It seems that at least for some problems actually finding a solution is considerably more difficult that checking a given solution. The example of checking whether a number can be written as a non-trivial product, or equivalently, that it is not prime, should be taken with a grain of salt, as in fact it has turned out that checking whether a number is a prime number can be done in polynomial time and therefore, from the perspective of polynomial-time computability, solving the problem and checking a given solution are equally difficult [AKS04].

However, it is unknown, and in fact one of the most important open questions in the whole of computer science whether it holds that $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$. It is widely believed that $\mathcal{P} \neq \mathcal{NP}$ and that therefore problems exist for which checking a solution is considerably simpler than computing one. One such problem, which belongs to $\mathcal{NP}$ but is not known to be in $\mathcal{P}$ is the problem Satisfiability or Sat for short. A Boolean CNF formula over a set $\{x_1, \ldots, x_n\}$ of variables is a conjunction of a set $\{C_1, \ldots, C_m\}$ of clauses, where each C_i is a disjunction of *literals*, that is, positive or negative occurrences of variables. The problem Satisfiability asks whether a given boolean formula φ admits a satisfying truth assignment. The problem clearly belongs to $\mathcal{NP}$ as φ can be evaluated for a given truth assignment in polynomial (even linear) time. It is, however, not known whether Sat $\in \mathcal{P}$.

Polynomial-Time Reductions and NP-Completeness. Let Π and Π' be two problems and let I be an instance of Π. A *polynomial-time reduction* of Π to Π' is a polynomial-time algorithm $\mathcal{A}$ that computes for each instance I of Π an instance $\mathcal{A}(I)$ of Π' with the property that I is a Yes-instance of Π if and only if $\mathcal{A}(I)$ is a Yes-instance of Π'. If there exists a polynomial-time reduction from Π to Π', we write $\Pi \lesssim_P \Pi'$. The motivation for this notation is that by using the reduction algorithm, any algorithm for solving Π' efficiently, immediately implies an algorithm for solving Π, and hence Π' is at least as hard as Π. The binary relation defined by $\lesssim_P$ on $\mathcal{NP}$ is reflexive and transitive and hence a quasi-ordering on $\mathcal{NP}$, which roughly orders the problems in $\mathcal{NP}$ according to their difficulty, neglecting differences in complexity that are polynomially bounded. For resolving the $\mathcal{P}$ vs. $\mathcal{NP}$ problem, the minimal and maximal elements of this relation are of special interest, as they are most likely to be separable by a super-polynomial complexity barrier. On the other hand, if a maximal element could be solved in polynomial-time this would

show the polynomial-time solvability of a large part of $\mathcal{NP}$ (and in fact that $\mathcal{P} = \mathcal{NP}$, as we will see in a moment).

For the minimal elements, note that for all problems Π and Π' in $\mathcal{P}$ we have $\Pi \lesssim_P \Pi'$ and therefore the set of minimal elements of $\lesssim_P$ is exactly $\mathcal{P}$. The maximal elements are more complicated and their existence is not at all obvious.

A problem Π is *NP-hard*, if every problem in $\mathcal{NP}$ can be reduced to Π in polynomial time, that is $\Pi' \lesssim_P \Pi$ for all $\Pi' \in NP$. A problem Π is *NP-complete* if it is NP-hard and $\Pi \in \mathcal{NP}$. Clearly, the NP-complete problems are the maximal elements of $\lesssim_P$ and every problem in $\mathcal{NP}$ can be reduced in polynomial-time to any NP-complete problem, making NP-complete problems the "most difficult" problems in $\mathcal{NP}$. If any of these problems admits a polynomial-time algorithm, then, by virtue of these reductions, all problems in $\mathcal{NP}$ admit polynomial-time algorithms and this would prove $\mathcal{P} = \mathcal{NP}$. On the other hand, if $\mathcal{P} \neq \mathcal{NP}$, then the NP-complete problems are the primary candidates for proving the non-existence of an efficient algorithm.

To prove that a problem Π is NP-hard it is sufficient to show that $\Pi' \lesssim_P \Pi$ for some known NP-complete problem Π', that is giving a polynomial time reduction from the NP-complete problem Π' to Π. Since NP-complete problems are the most difficult problems in $\mathcal{NP}$, the widely believed assumption that $\mathcal{P} \neq NP$ implies that these problems are unlikely to be efficiently solvable.

There is, however, a subtle problem. Namely, it is not clear that such problems even exist. In his seminal paper Cook [Coo71] showed that Sat, the problem of deciding whether a given boolean formula in CNF is satisfiable is NP-complete. His proof showing explicitly that every problem instance of every problem in $\mathcal{NP}$ can be encoded as a Sat formula of polynomial size was a major breakthrough in complexity theory. Moreover, it lay the ground for proving further problems NP-complete; Karp [Kar72] presented a list of 21 NP-complete problems, in particular combinatorial and graph theoretical problems. Today, a vast number of problems are known to be NP-complete.

Basic NP-complete Problems. In this thesis we will prove some problems to be NP-complete, in order to show that they are unlikely to be polynomial-time solvable. We now describe three NP-complete problems that we use in our reductions. Two of these problems are variants of problems in Karp's initial list of 21 NP-complete problems.

The first is a variant of Sat with additional constraints. The problem k-Sat is a special case of Sat, where the formulas are restricted to clauses with at most k literals. This problem is also NP-complete for $k \geq 3$ [GJ79], while 2-Sat is polynomial-time solvable [APT79]. Given a formula φ we can associate with φ a graph representing the interaction of variables and clauses in the formula, the *variable–clause graph* G_φ whose vertices are the variables and clauses of φ. A variable and a clause are connected by an edge in G_φ if the variable occurs in the clause. The problem Planar 3-Sat asks whether a given 3-Sat formula whose variable–clause graph is planar admits a satisfying truth assignment. Planar 3-Sat is NP-hard [Lic82]. Due to the planarity constraint on the variable–clause graph, Planar 3-Sat is especially well-suited for proving NP-hardness of problems involving planarity for both graph-theoretic and geometric problems.

The second problem we will use is HamiltonianCircuit. The problem asks whether a given graph $G = (V, E)$ contains simple cycle that visits all vertices. It is NP-hard for directed as well as for undirected graphs, even if the input graph is planar and has maximum degree 3 [GJ79, Ple79].

Finally, we will use the problem SteinerTree. An instance of SteinerTree consists of an undirected graph $G = (V, E)$ and set $T \subseteq V$ of *terminals*. It asks, for the smallest

subgraph of G (in terms of number of edges) that is connected and contains all terminals. More precisely, we use the decision version of this problem, which has an additional integer input parameter k and asks whether a subgraph with k edges suffices to connect all the terminals. Again this problem is NP-complete [GJ79].

Part I.

Combinatorial Optimization on Planar Graphs

Chapter 3

Augmenting the Connectivity of Planar and Geometric Graphs

In this chapter we study connectivity augmentation problems. Given a connected graph G with some desirable property, we want to make G 2-vertex connected (or 2-edge connected) by adding edges such that the resulting graph keeps the property. The aim is to add as few edges as possible. The property that we consider is planarity, both in an abstract graph-theoretic and in a geometric setting, where vertices correspond to points in the plane and edges to straight-line segments.

We show that it is NP-hard to find a minimum-cardinality augmentation that makes a planar graph 2-edge connected. For making a planar graph 2-vertex connected this was known. We further show that both problems are hard in the geometric setting, even when restricted to trees. The problems remain hard for higher degrees of connectivity. On the other hand we give polynomial-time algorithms for the special case of convex geometric graphs.

We also study the following related problem. Given a planar (plane geometric) graph G, two vertices s and t of G, and an integer c, how many edges have to be added to G such that G is still planar (plane geometric) and contains c edge- (or vertex-) disjoint s–t paths? For the planar case we give a linear-time algorithm for $c = 2$. For the plane geometric case we give optimal worst-case bounds for $c = 2$; for $c = 3$ we characterize the cases that have a solution.

The chapter is based on joint work with Alexander Wolff [RW08a, RW08b].

3.1. Introduction

Augmenting a given graph to increase its connectivity is important, for example, for making communication networks resistant against node and link failures. The planar version of the problem, where the augmentation has to preserve planarity, also has applications in graph drawing [KB91]. Many graph-drawing algorithms guarantee nice properties (such as convex faces) for graphs with high connectivity. To apply such an algorithm to a less highly connected graph, one adds edges until one reaches the required level of connectivity, uses the algorithm to produce the drawing, and finally removes edges that were added before. With each removal of an edge, however, one might lose some of the nice properties (such as the convexity of a face). Hence, it is natural to look for an augmentation that uses as few edges as possible. Recall that a graph is *c-vertex connected* (or simply *c-connected*)

if the removal of any subset of $c-1$ vertices does not disconnect the graph. Analogously, a graph is *c-edge connected* if the removal of any subset of $c-1$ edges does not disconnect the graph. It is common to use the term *biconnected* for 2-vertex connected and the term *bridge-connected* for 2-edge connected.

In this chapter, we consider the following two problems.

Planar 2-Vertex Connectivity Augmentation (PVCA):

> Given a connected planar graph $G = (V, E)$, find a smallest set E' of vertex pairs such that the graph $G' = (V, E \cup E')$ is planar and biconnected.

Planar 2-Edge Connectivity Augmentation (PECA)

> is defined as PVCA, but with *biconnected* replaced by *bridge-connected*.

Related Work. The corresponding problems without the planarity constraints have a long history, both for directed and undirected graphs. Eswaran and Tarjan [ET76] showed that the unweighted cases can be solved in polynomial time, whereas the weighted versions are hard. Frederickson and Ja'Ja' [FJ81] gave $O(n^2)$-time factor-2 approximations and showed that augmenting a directed acyclic graph to be strongly connected, and augmenting a tree to be bridge- or biconnected, is NP-complete—even if weights are restricted to the set $\{1, 2\}$. Hsu [Hsu02] gave an $O(m+n)$-time sequential algorithm for (unit-weight) 2-vertex connectivity augmentation that can be parallelized well.

Kant and Bodlaender [KB91] showed that PVCA is NP-complete and gave 2-approximations for both PVCA and PECA that run in $O(n \log n)$ time. Their 1.5-approximation for PVCA turned out to be wrong [FM98]. Fialko and Mutzel [FM98] gave a 5/3-approximation for PVCA. Kant [Kan96] showed that PVCA and PECA can be solved in linear time for outerplanar graphs.

Provan and Burk [PB99] considered related problems. Given a planar graph $G = (V, E_G)$ and a planar biconnected (bridge-connected) graph $H = (V, E_H)$ with $E_G \subseteq E_H$, find a smallest set $E' \subseteq E_H$ such that $G' = (V, E_G \cup E')$ is planar and biconnected (bridge-connected). They show that both problems are NP-hard if G is not necessarily connected and give $O(n^4)$-time algorithms for the connected cases.

We also consider a geometric version of the above problems. Recall that a *geometric* graph is a graph where each vertex v corresponds to a point $\mu(v)$ in the plane and where each edge uv corresponds to the straight-line segment $\overline{\mu(u)\mu(v)}$ connecting u and v. We are exclusively interested in *plane* geometric graphs, that is, geometric graphs whose edges neither cross each other nor contain vertices other than their endpoints. Therefore, in this chapter by geometric graph we always mean a plane geometric graph. Given a geometric graph G we again want to find a (small) set of vertex pairs such that adding the corresponding edges to G leaves G plane and augments its connectivity.

In this context, Rappaport [Rap89] showed that it is NP-complete to decide whether a set of line segments can be connected to a simple polygon, that is, geometric PVCA and PECA are NP-complete. Abellanas et al. [AGH+08] gave worst-case bounds for geometric PVCA and PECA. For geometric PVCA, they showed that $n-2$ edges are sometimes needed and are always sufficient. For geometric PECA, they proved that $2n/3$ edges are sometimes needed and $6n/7$ edges are always sufficient for graphs with n vertices. In the special case of plane geometric trees (with n vertices) they show that $n/2$ edges are sometimes needed and that $2n/3$ edges are always sufficient for PECA. Tóth [Tót08] lowered the upper bounds to $\lfloor n/2 \rfloor$ for n-vertex trees and $2n/3 + O(1)$ for arbitrary n-vertex plane geometric graphs. Tóth and Valtr [TV09] characterized all cases

where a 2-vertex or a 2-edge connected plane geometric graph can be augmented to be 3-vertex or 3-edge connected. They further showed that in both cases $n-2$ new edges suffice and are sometimes necessary for graphs with n vertices. Al-Jubeh et al. [AJIR+09] proved that an *arbitrary* (not necessarily 2-edge connected) 3-edge augmentable plane geometric graph with n vertices can be 3-edge augmented with $2n-2$ new edges and that this number is sometimes necessary. Their augmentation algorithm runs in $O(n \log^2 n)$ time.

Contribution. First we show that PECA is NP-complete, too. This answers an open question posed by Kant [Kan93].

Second, we sharpen the result of Rappaport [Rap89] by showing that geometric PVCA and PECA are NP-complete even if restricted to trees. Not unexpectedly, the problems remain hard for higher degrees of connectivity: finding a minimum-cardinality augmentation that makes a plane geometric $(c-1)$-vertex connected graph c-vertex connected is also NP-hard for $c = 3, \ldots, 5$. The gadgets in our construction are such that they establish hardness for both vertex and edge connectivity. Recall that any planar graph has a vertex of degree at most 5 and hence is at most 5-connected.

Third, we give algorithms that solve geometric PVCA and PECA in polynomial time for *convex* geometric graphs, that is, graphs whose vertex sets correspond to point sets in convex position.

Table 3.1 gives an overview about what is currently known about the complexity of PVCA and PECA and their geometric variants.

problem	planar	outerplanar	geometric	convex
PVCA	NPC [KB91]	$O(n)$ [Kan96]	NPC	$O(n)$
PECA	NPC	$O(n)$ [Kan96]	NPC	$O(n)$
weighted PVCA	NPC	open	NPC	$O(n)$
weighted PECA	NPC	open	NPC	$O(n^2)$

Table 3.1.: Complexity of various versions of PVCA and PECA. NPC stands for NP-complete.

Fourth, we consider a related problem, the geometric s–t path augmentation problem. Given a plane geometric graph G, two vertices s and t of G, and an integer $c > 0$, is it possible to augment G such that it contains c edge-disjoint (c vertex-disjoint) s–t paths? We restrict ourselves to $c \in \{2, 3\}$. For $c = 2$ we show that edge-disjoint s–t path augmentation can always be done and needs at most $n/2$ edges, where n is the number of vertices in the graph G. We give an algorithm that computes such an augmentation in linear time. The tree that yields the above-mentioned lower-bound of Abellanas et al. [AGH+08] also shows that our bound is tight. For $c = 3$ we show that edge-disjoint s–t path augmentation is always possible, and we give an $O(n^2)$-time algorithm that decides whether a given graph has a vertex-disjoint s–t path augmentation.

In this chapter we use the term *leaf* for a degree-1 vertex in any graph, not only in a tree.

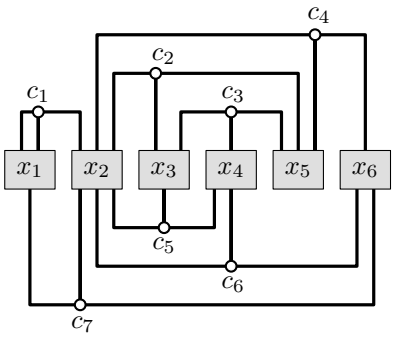


Figure 3.1.: Layout of the variable–clause graph corresponding to a planar 3-SAT formula with variables $x_1, \ldots, x_6$ and clauses $c_1, \ldots, c_7$.

3.2. Complexity

In this section we show that PECA is NP-complete. This settles an open problem posed by Kant [KB91]. Kant proved that the minimum biconnectivity augmentation problem is NP-complete and gave 2-approximations for both problems, PVCA and PECA [KB91]. We also strengthen the result of Rappaport [Rap89] and show that geometric PECA and geometric PVCA are NP-complete even in the case of trees. We also show hardness for the corresponding problems with higher degrees of connectivity.

3.2.1. Complexity of PECA

We start by settling the complexity of PECA. For our proof, recall that an *embedding* of a connected planar graph is given by a circular ordering of the incident edges around each vertex.

Theorem 3.1. PECA *is NP-complete.*

Proof. PECA is in $\mathcal{NP}$ since given a planar graph G and an integer $k > 0$ we can guess (with positive probability) a set E' of at most k non-edges of G and then test efficiently whether $G + E'$ is planar. We prove the NP-hardness of PECA by reducing from the problem Planar3SAT, which is known to be NP-hard [Lic82]. An instance of Planar3SAT is a 3SAT formula φ whose variable–clause graph is planar. Such a graph can be laid out (in polynomial time) such that variables correspond to pairwise disjoint axis-parallel rectangles intersecting a horizontal line and clauses correspond to non-crossing three-legged "combs" above or below that line [KR92], see Figure 3.1.

Note that if a graph G has k leaves, at least $k/2$ edges need to be added to bridge-connect the graph. In case $k/2$ edges suffice, each of these edges connects two leaves and no two edges are incident to the same leaf. In other words, the edges form a perfect matching of the leaves. We now construct a planar graph G_φ that can be augmented with a perfect leaf matching if and only if φ is satisfiable. The graph G_φ consists of so-called *gadgets*, that is, subgraphs that represent the variables, literals, and clauses of φ, see Figure 3.2. For each gadget, we will argue that there are only a few ways to embed and augment it with a perfect leaf matching. Note that our construction connects variable gadgets corresponding to neighboring variables in the layout of the variable–clause graph of φ. Hence G_φ is always connected. Additionally, we identify the left boundary of the leftmost variable gadget with the right boundary of the rightmost variable gadget.

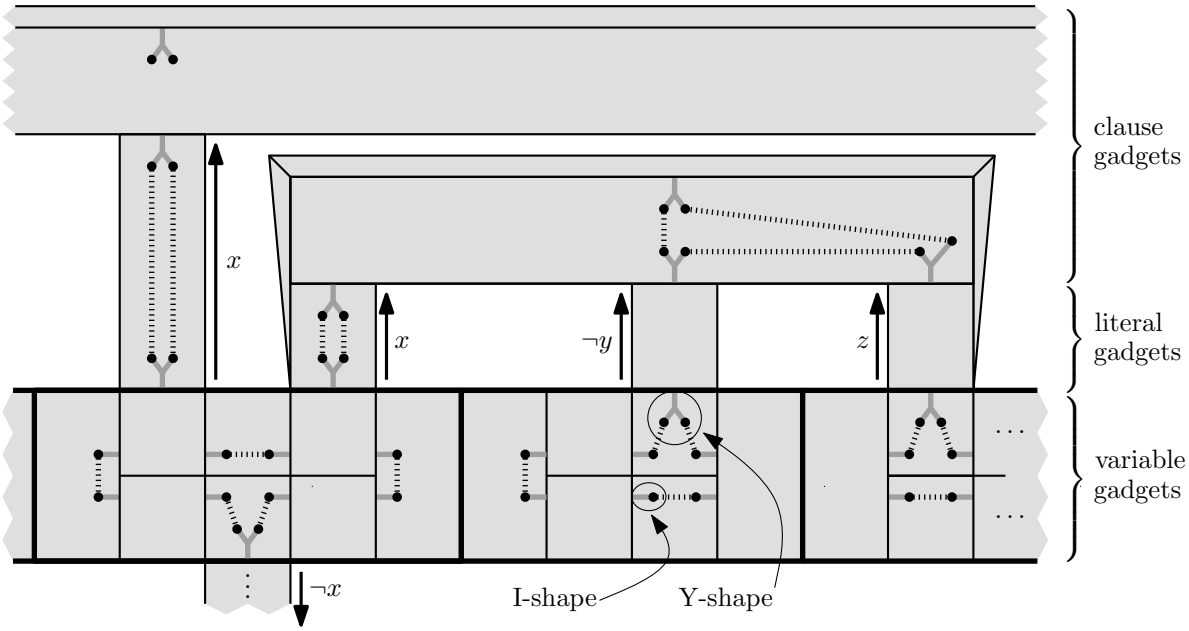


Figure 3.2.: Part of the graph G_φ for a 3SAT formula φ that contains the clause $(x \wedge \neg y \wedge z)$. The augmentation (dotted edges) corresponds to the assignment $x = y = \mathit{false}$ and $z = \mathit{true}$.

In the figure, leaves are highlighted by small black disks. All bends and junctions of line segments represent vertices of degree greater 1. The (black and dark gray) solid line segments between adjacent vertices represent the edges of G_φ; the thick dotted line segments represent non-edges of G_φ that are candidates for an augmentation of G_φ. The set of black solid edges forms a subgraph of G_φ that we call the *frame*. The dark gray solid edges form what we call *I-shapes* and *Y-shapes*, which connect singles leaves and pairs of leaves to the frame, respectively. In Figure 3.2, we marked examples of I- and Y-shapes.

Consider the graph G'_φ that we obtain from the frame by contracting all vertices of degree 2. We claim that G'_φ is 3-vertex connected. This is true since (a) the subgraph of G'_φ induced by the variable gadgets is 3-connected and (b) each subgraph induced by a clause gadget and the three corresponding literal gadgets is also 3-connected and is attached to the former (variable gadget) subgraph in six vertices.

Recall a classical result of Whitney's [Whi32] which says that a planar graph has a unique dual (and thus a unique embedding) if and only if it is 3-vertex connected. Hence, G'_φ has a unique embedding. The same holds for the frame of G_φ as it is a subdivision of G'_φ. In other words, the embedding of G_φ is fixed up to the embedding of the I- and Y-shapes.

We say that an augmentation of a gadget or of G_φ is *tight* if the new edges form a leaf matching. It is easy to see that if G_φ has a tight augmentation, then G_φ has an embedding such that the following two properties hold.

(P1) Each face contains an even number of leaves.

(P2) Each face that contains a Y-shape contains at least four leaves.

Note that in a tight augmentation, the two leaves of a Y-shape cannot be matched to each other since the edge of the Y-shape that is not incident to a leaf would be a bridge.

Our variable gadget consists of two rows of square faces where the horizontal edge between the two leftmost faces and the horizontal edge between the two rightmost faces is missing. Effectively, the faces of a variable gadget form a cycle. Starting from the leftmost (rectangular) face, we call the faces *odd* and *even*. To every interior vertical edge an I-shape is attached. Due to (P1), the I-shapes can be matched in exactly two ways; either in the odd or in the even faces. If the matching is in the even faces, then the corresponding variable is true, and vice versa.

A literal gadget consists of a square face that lies immediately above or below the variable gadget. A positive literal (such as the ones labeled with x in Figure 3.2) is attached to an even face, a negated literal (such as the one labeled with $\neg y$ in Figure 3.2) is attached to an odd faces. A literal gadget contains two Y-shapes, one attached to each of its two horizontal edges. Due to (P2) these Y-shapes are embedded either both inside or both outside the literal gadget. Again due to (P2) the Y-shapes must be embedded inside the literal gadget if no I-shapes are embedded into the adjacent face of the variable gadget. In this case, the literal has the value *false*. If two I-shapes are embedded into the adjacent face of the variable gadget, the Y-shapes of the literal gadget can (but don't have to) be embedded to the outside (see the literal $\neg y$).

Finally, each clause gadget consists of a single rectangular face that contains a Y-shape. If G_φ has a tight augmentation, then, due to (P2), at least two other leaves are embedded into every clause gadget face. This means that for each clause gadget, the Y-shapes of at least one adjacent literal gadget are embedded to the outside. In other words, at least one of the literals that make the clause is *true*. Hence, φ has a satisfying truth assignment.

Conversely, it is easy to see that if φ has a satisfying truth assignment, then all gadgets have a tight augmentation and hence, so does G_φ.

We use a constant number of vertices and edges for each literal and clause gadget, thus our reduction—including the computation of the embedding of the variable–clause graph—is polynomial. □

Note that the graph constructed in the proof is 2-edge connected if and only if it is biconnected. Hence our proof also shows that PVCA is NP-complete.

3.2.2. Geometric PVCA and Geometric PECA

Next we show that geometric PVCA and geometric PECA are NP-complete as well. With a simple modification it follows that the problems are even NP-complete if the input is restricted to plane geometric trees. With another modification, we show hardness of the corresponding problems for higher degrees of connectivity.

Note that we cannot recycle our proof of Theorem 3.1 to show hardness of the geometric variants of the problems: there, we exploited that certain parts of the graph (the I- and Y-shapes) could be embedded in different (but adjacent) faces. Here, we are given embedded graphs; we cannot even move vertices or edges. To show hardness, we exploit this rigidity. Our proof is again by reduction from Planar3SAT. Although the graph that we are about to construct looks very different from the one we constructed in the proof of Theorem 3.1, similar functional units (as the I- and Y-shapes) will play a role.

Theorem 3.2. *Let G be a connected plane geometric graph with k leaves. It is NP-complete to decide whether it is possible to augment G with $k/2$ edges such that G becomes bridge- or biconnected.*

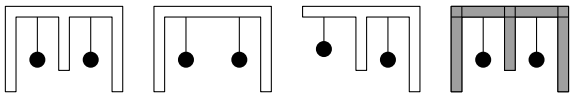


Figure 3.3.: Various variants of loopholes, (a)–(c), and subdivision of the boundary of a loophole into rectangular block (d).

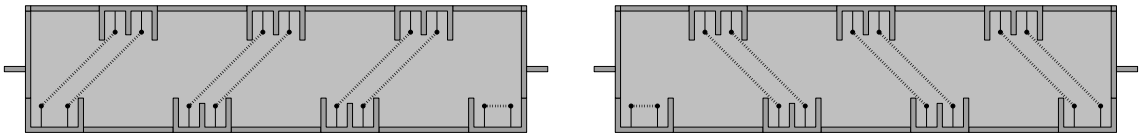


Figure 3.4.: Variable gadget without any adjacent literal gadgets; the two minimum augmentations correspond to the values of the variable.

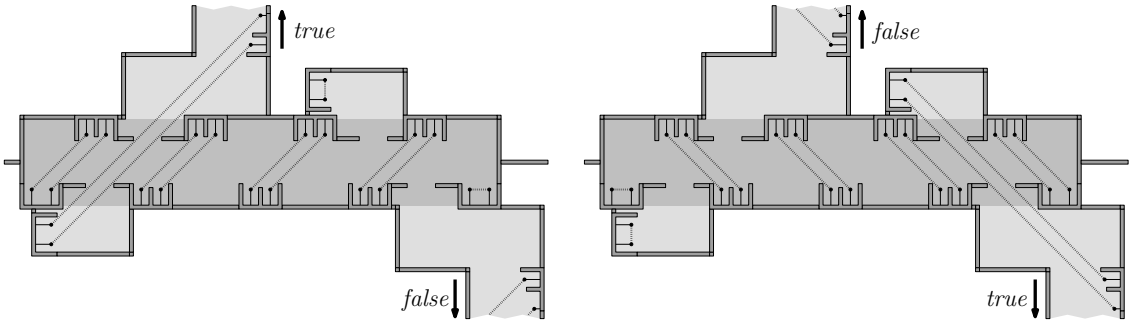
Proof. For membership in $\mathcal{NP}$ we argue as in the proof of Theorem 3.1. To show the NP-hardness of the two problems, we again reduce from Planar3SAT and construct a connected plane geometric graph G consisting of gadgets that represent the variables, literals, and clauses of the given planar 3SAT formula. Recall once more that we need to add at least $k/2$ edges in order to make G bridge- or biconnected since every leaf must lie on a cycle afterwards and must hence be incident to one of the added edges.

The basic building block of our gadgets is what we call a *loophole*. The default loophole, depicted in Figure 3.3a, consists of an E-shaped cycle and two attached I-shapes which are placed such that their leaves cannot see each other. (Recall that an I-shape is a leaf with its incident edge.) In contrast, in a *self-connecting* loophole (see Figure 3.3b), the two leaves do see each other. A *skewed* loophole is a loophole that misses one of the boundaries (Figure 3.3c). In the terminology of the proof of Theorem 3.1, a default loophole corresponds to a Y-shape, and a self-connecting loophole corresponds to a pair of I-shapes in the same face. Skewed loopholes are similar to default loopholes; their shape differs to allow for certain connections without crossings.

Again, all our gadgets are surrounded by *walls*, that is, by biconnected subgraphs that ensure that the whole construction without the leaves is biconnected. In the figures, walls are indicated by gray rectilinear polygons.

We are now ready to describe our variable gadget, see Figure 3.4. It consists of two parallel rows of evenly spaced loopholes with the upper loopholes pointing downwards and the lower ones pointing upwards. The lower row contains one loophole more than the upper one and its two outermost loopholes are self-connecting. The rows are aligned so that every loophole (except the first and last of the lower row) lies horizontally between two opposing loopholes—which we call its partners—on the other row. The distances between the loopholes and the two rows are chosen such that the I-shapes of each loophole can only be connected to the leaves of its two partners on the other row without producing a crossing. As in the proof of Theorem 3.1, we connect neighboring variable gadgets to ensure that the resulting graph is connected.

For any minimum augmentation, the two I-shapes of a loophole on the upper row must be connected to the two I-shapes of its left or right partner on the other row and the edges in the augmentation have slope 1 or -1. By construction this choice has to be the



(a) state *true* (b) state *false*

Figure 3.5.: Variable gadget (middle gray) with two adjacent literal gadgets (light gray). The upper left literal gadget transmits the logical value of the variable, whereas the lower right literal gadget transmits the negation of that value.

same for each loophole on the upper row, otherwise crossings would occur. On the lower row, depending on the choice either the first or last loophole does not receive new edges and its I-shapes must be connected. Hence, a variable has exactly two different minimum augmentations. The two possible states are shown in Figure 3.4. We say that the variable is in state *true* if the edges connecting loophole partners have slope 1 and *false* if they have slope -1.

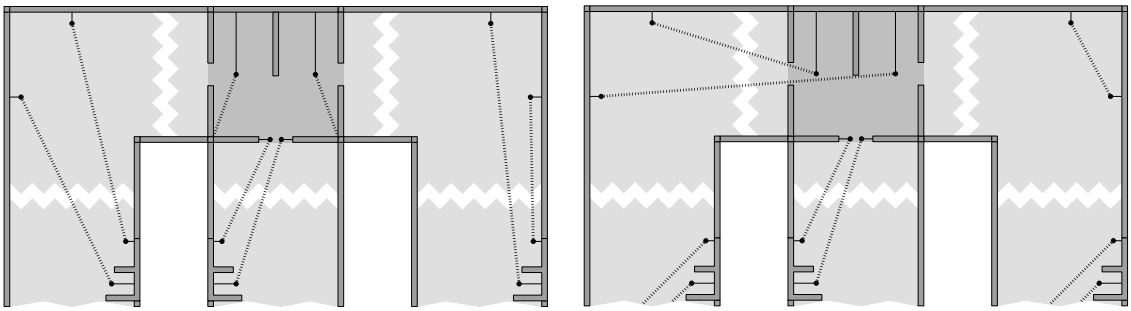
Next we show how the state of a variable is transmitted to the gadgets that represent the clauses in which the variable occurs. This is the job of the literal gadgets. Roughly speaking, for each literal gadget we remove two wall pieces of the variable gadget, and attach a self-connecting loophole on one side and a skewed loophole on the other side, see Figure 3.5.

If the literal is positive (as in the case of the upper left literal gadget in Figure 3.5), a leaf in one of the two loopholes can be connected to a leaf in the other loophole if and only if the new edges in the variable gadget all have slope 1. If the literal is negated (as in the case of the lower right literal gadget in Figure 3.5), leaves of the two loopholes can only be interconnected if the new edges in the variable gadget all have slope -1.

In this way, the leaves of the skewed loophole can be matched to the leaves of the corresponding self-connecting loophole if and only if the value of the variable satisfies the corresponding literal. Otherwise, leaves of the skewed loophole have to be connected to a vertex inside the clause gadget, which we present next.

The clause gadget consists of a square that contains a loophole and two L-shaped wall parts that occupy the corners opposite of the loophole, see Figure 3.6. On three sides, the square is connected via literal gadgets to the gadgets of the three variables that form the clause in the given planar 3SAT formula. Each literal gadget contains two I-shapes that are positioned such that they see (a) each other, (b) the two I-shapes inside the square, and (c) the two I-shapes of the skewed loophole where the literal gadget is attached to a variable. It is not hard to see that if any of the skewed loopholes is matched to leaves in the variable gadget, then the two I-shapes in the literal gadget are free to connect to the two I-shapes in the square. Only if all three skewed loopholes are matched to I-shapes in their literal gadget, then the two I-shapes in the center square require an additional edge.

As in the proof of Theorem 3.1 we use a constant number of vertices and edges for each literal and clause gadget, thus our reduction—including the computation of the embedding of the variable–clause graph—is polynomial. □



(a) state *false* (b) state *true*

Figure 3.6.: Clause gadget (middle gray) with the three adjacent literal gadgets (light gray). If all literals are *false*, the leaves in the clause gadget (or some other leaves) must be matched to non-leaves, or a leaf receives more than one new edge (a). If at least one literal is *true*, there is an augmentation that matches all leaves to other leaves. The right figure (b) depicts the situation where the literals corresponding to the left and the right gadget are true and the literal corresponding to the middle gadget is false.

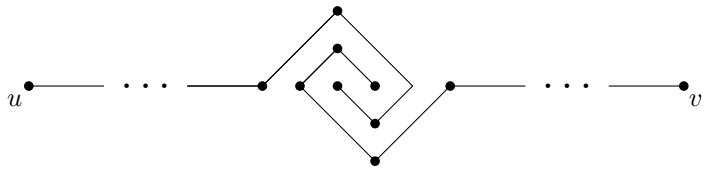


Figure 3.7.: Construction that removes a cycle, locally leaves only one possibility to augment, and does not interfere with the remainder of the graph with respect to augmentation.

By a simple trick we can slightly strengthen the result of Theorem 3.2.

Corollary 3.1. *It is NP-complete to decide whether a plane geometric tree with k leaves can be augmented to be bridge- or biconnected with $k/2$ edges.*

Proof. The proof is by reduction from the previous case. Let G be a connected plane geometric graph with k_G leaves. We now show how to remove cycles from G. Since the construction leaves G connected, the resulting graph is a tree.

To reduce the number of cycles we replace an arbitrary edge that lies on a cycle by the construction shown in Figure 3.7. Note that we can make the spiral in the center of the construction so small that is does not prevent any connections in the remainder of the graph. We iterate this construction until there are no cycles left. The resulting graph is a tree T. Let k_T be the number of leaves of T. It is clear that for each of the new leaves (in the spiral centers) there is only one way to connect to another leaf, namely to the one that restores the cycle we removed before. Hence, T can be augmented with $k_T/2$ edges if and only if G can be augmented with $k_G/2$ edges.

The reduction can be performed in polynomial time since we introduce at most one spiral per edge of G, each consisting of a constant number of edges. The length of the shortest new edge of T is roughly proportional to the smallest distance among the vertices of G. □

We now generalize the proof of Theorem 3.2 to show that for any $2 \leq c \leq 5$, it is NP-hard to augment a plane geometric graph to be c-connected by adding a given number of edges. Note that any planar graph has a vertex of degree at most 5, so planar graphs

Figure 3.8.: Adding edges to leaves in the second step of the construction of G_φ (here with $c = 5$).

are at most 5-connected. To show that our construction has the desired properties, let us make some simple observations.

Observation 3.1. Let G be a graph with vertices u and v. Then the following properties hold:

(i) If $G - u$ is c-connected and u has degree at least c, then G is c-connected as well.

(ii) If $G - \{u, v\}$ is c-connected and vertices u and v are not adjacent and have degree at least $c - 1$ but no common neighbor, then $G + uv$ is c-connected.

Proof. For showing property (i), suppose that S is a separator of G with $|S| < k$. Since S is no separator in $G - u$, S splits off only u from G. Hence, S contains all neighbors of u, and thus $|S| \geq k$. Contradiction.

Similarly, for property (ii), suppose that S is a separator of $G+uv$ with $|S| < k$. Since S is not a separator of $G-\{u, v\}$, S splits off u or v (or both) in $G+uv$. This, however, is not possible since both u and v have degree at least c in $G + uv$ and since their neighborhoods are disjoint. □

To generalize the proof of Theorem 3.2 to higher degrees of connectivity, we make the graph G_φ $(c-1)$-connected in two steps.

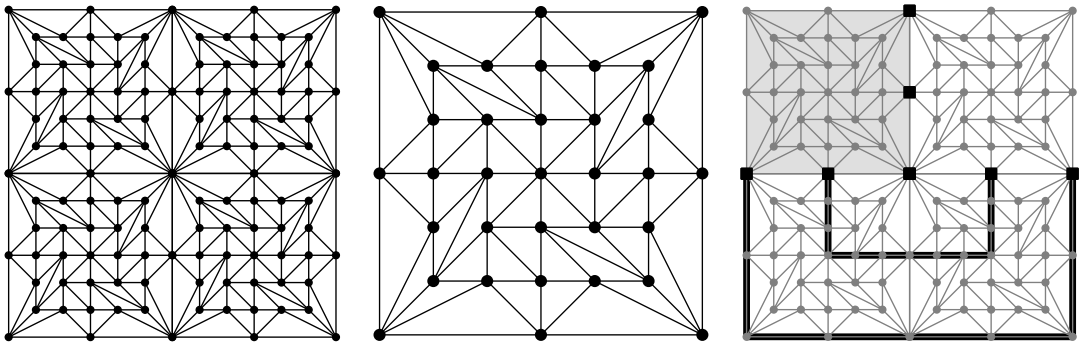
First, we temporarily remove all I-shapes from G_φ, subdivide the walls of the loopholes as in Figure 3.3d into gray rectangles, and replace each gray rectangle in Figures 3.5 and 3.6 by a copy of the graph depicted in Figure 3.9a. We stick two building blocks together by identifying the five vertices on the edge of one block to the five corresponding vertices on the edge of the other block. Call the resulting graph G^1_φ.

Second, we treat the former I-shapes of G_φ. We connect each leaf of G_φ by $(c-2)$ additional edges to the boundary of G^1_φ such that no two leaves have a common neighbor, see Figure 3.8. We call the resulting graph G^2_φ. We now show that G^2_φ does the job.

Theorem 3.3. *It is NP-hard to decide the following question: given integers $2 \leq c \leq 5$ and $k \geq 1$ and a $(c-1)$-connected plane geometric graph G, can G be augmented to being c-connected by adding at most k edges?*

Proof. We again reduce from Planar3SAT, along the lines of the proof of Theorem 3.2. We first show that G^2_φ is $(c-1)$-connected.

In order to see this, we claim that G^1_φ is 5-connected. The walls of G^1_φ are made from copies of our basic building blocks. Such a block is 5-connected for two reasons; (a) it consists of four copies of the smaller 5-connected graph, a *sub-block*, depicted in Figure 3.9b, whose 5-connectivity we have verified by a computer program and (b) two neighboring sub-blocks lie in the same 5-connected component. To see (b), consider the five *portals* that we define on the boundary of each sub-block, see the black squares in Figure 3.9c.



(a) (b) (c)

Figure 3.9.: A building block (a), a 5-connected sub-block with 33 vertices (b), proof that a building block is 5-vertex connected (c).

Each vertex in a sub-block has five vertex-disjoint paths to its portals, which are connected to the corresponding portals of the neighboring sub-block via (possibly trivial) pairwise vertex-disjoint paths. Observations (a) and (b) plus symmetry show our claim.

Given that G_φ^1 is 5-connected and $c \leq 5$, property (i) of Observation 3.1 yields that G_φ^2 is $(c-1)$-connected. Note that the leaves of G_φ and the degree-$(c-1)$ vertices of G_φ^2 are in one-to-one correspondence. Let K be their number. Clearly, in order to make G_φ^2 c-connected, we need at least $K/2$ new edges. We claim that the graph G_φ^2 that we have constructed above can be made c-connected by adding $K/2$ edges if and only if G_φ can be made biconnected by adding $K/2$ edges.

If G_φ^2 can be made c-connected by adding $K/2$ edges, then these edges form a matching of the vertices of degree $c-1$. This matching can also be added (in a plane fashion) to G_φ.

Now we turn to the other direction. If G_φ can be made biconnected by adding $K/2$ edges, we add the corresponding edges to G_φ^2. We have shown above that G_φ^1 is 5- and thus c-connected. Now property (ii) of Observation 3.1 yields that each of the remaining vertices lies in the same c-connected component as G_φ^1. This finishes the proof of our claim.

Clearly, our reduction is polynomial. □

Using the same graph G_φ^2 in the reduction, we can prove the statement for edge connectivity, too.

Corollary 3.2. *Given integers $2 \leq c \leq 5$ and $k \geq 1$ and a $(c-1)$-edge connected plane geometric graph G, it is NP-hard to decide whether G can be augmented to being c-edge connected by adding at most k edges.*

Figure 3.10.: A cycle (left) and a near-cycle (right).

3.3. Convex Geometric Graphs

In this section we show that geometric PVCA and geometric PECA can be solved in polynomial time for connected *convex* geometric graphs, that is, for graphs whose vertices are in convex position. We focus on augmenting a given connected convex geometric graph to bridge- and biconnectivity. Note that every convex geometric graph is outerplanar and hence contains a vertex of degree at most 2, which prevents higher connectivity.

We first consider the very simple problem of biconnecting a convex geometric graph, see Section 3.3.1. Then we give an algorithm that computes an edge set of minimum cardinality that bridge-connects a convex geometric graph, see Section 3.3.2. Finally, in Section 3.3.3 we consider a weighted version of bridge-connectivity augmentation. We give an algorithm that computes a minimum-weight augmentation in a connected n-vertex convex geometric graph in $O(n^2)$ time.

We assume that for a geometric graph the edges incident to a vertex are ordered clockwise. If this information is not provided, we can easily compute it in $O(n \log n)$ time.

3.3.1. Biconnecting Convex Geometric Graphs

Consider an arbitrary connected convex geometric graph G. Suppose that there are two consecutive vertices u and v on the convex hull that are not connected by an edge. Since G is connected, adding the edge uv creates a new face F. It is not hard to see that every vertex of $F - \{u, v\}$ disconnects G. Hence, in a biconnected convex graph all edges of the convex hull must be present.

On the other hand if all edges of the convex hull are present, then the graph is biconnected. Hence, it suffices to add all edges of the convex hull that are not already in G to make G biconnected. This is also the minimum number of edges that must be added. As the convex hull of the point set can be computed in linear time if G is connected [Mel87], this can be done in linear time.

3.3.2. Bridge-Connecting Convex Geometric Graphs

In this section we consider the problem of bridge-connecting a convex geometric graph $G = (V, E)$. We start by considering two basic graphs that are especially easy to bridge-connect, the *cycle* and the *near-cycle* shown in Figure 3.10. While the cycle is already bridge-connected, the near-cycle is not. It can, however, be bridge-connected by adding the single missing edge to form a cycle.

The basic idea is to decompose an arbitrary convex geometric graph into cycles and near-cycles and to use this decomposition to compute in a greedy fashion an edge set of minimum cardinality that bridge-connects the graph.

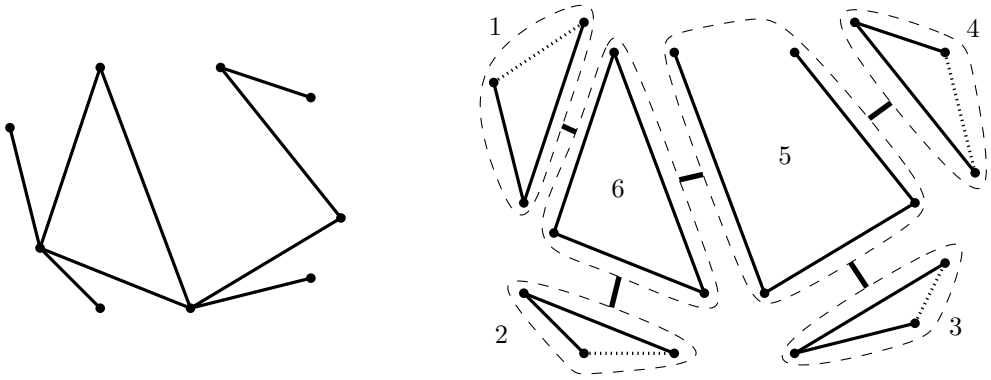


Figure 3.11.: A convex geometric graph (left) and its decomposition along interior edges (right). The dashed edges form a bridge-connectivity augmentation of minimal size, the numbers indicate the processing order of the components.

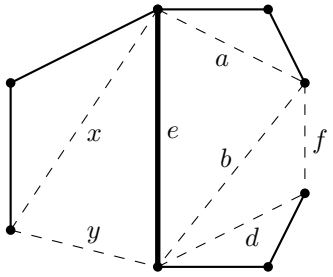
We differentiate between two types of edges. If an edge connects two consecutive vertices of the convex hull, we call it an *outer* edge, otherwise an *inner* edge. Note that if G is a connected convex geometric graph that does *not* contain an inner edge, then G is a cycle or a near-cycle.

Otherwise, an inner edge $e = uv$ can be used to split G into two subgraphs that can be augmented almost independently. The line defined by e splits the vertex set of G into two convex point sets P_1 and P_2. We then define, for $i = 1, 2$, the graph G_i as the subgraph of G induced by $P_i \cup \{u, v\}$. The interplay between augmentations of these two graphs is very limited since adding any edge between two vertices that are distinct from u and v and that do not belong to the same subgraph would introduce a crossing with e and is hence forbidden. On the other hand, the two augmentations are not completely independent as it suffices for e to be in *one* cycle. Hence, we store, for each edge e of G, a flag indicating whether e is already part of a cycle in the current partial augmentation. Initially all these flags are set to false.

Splitting G recursively along all inner edges defines a tree $\mathcal{T}$ whose nodes correspond to subgraphs of G. Two nodes of $\mathcal{T}$ are adjacent if and only if the corresponding subgraphs share an edge of G. Note that the tree $\mathcal{T}$ is essentially the weak dual of the graph that is obtained from G by adding all edges of the convex hull. The nodes of this tree correspond to components that are cycles or near-cycles. Starting from $E' = \emptyset$, we compute a minimum augmentation E' of G by iteratively augmenting a component C that corresponds to a leaf of $\mathcal{T}$. We do this as follows.

Let $e = uv$ be the edge that is shared by C and its parent in $\mathcal{T}$, and let $C' = C \setminus \{u, v\}$. We distinguish three different types of components. If C is a cycle, we mark all edges of C and remove C' from G. If C is a near-cycle that contains at least one edge except e that is not yet marked, we add to E' the unique edge that completes the cycle, mark e as lying on a cycle and remove C' from G. Finally, if C is a near-cycle and each edge except possibly e has been marked, we do not add any edge to E' and remove all vertices of C' from G. See Figure 3.11 for an example. Note that component 5 does not require an edge although it is not a cycle.

Once we have processed the last component of G, the set E' is an augmentation of G since we only remove edges from G that are marked as lying on cycles in $G + E'$. The minimality of E' follows from the fact that in each component we need to add at most one edge and we only add an edge to a component if it is strictly required.



edge	a	b	d	f
cost	2	4	5	10

Figure 3.12.: Example of a minimum-weight augmentation.

The algorithm can be implemented to run in linear time. The initial computation of $\mathcal{T}$ takes linear time, maintaining a list of leaves of the decomposition tree can be done in constant time per step and processing a component C takes time linear in the size of C.

We summarize our result.

Theorem 3.4. *Let S be a set of n points in the plane, and let $G = (S, E)$ be a connected convex geometric graph. There is an efficient algorithm that computes a minimum-cardinality set E' of edges such that $G + E'$ is bridge-connected. If the convex hull of S and the corresponding embedding of G is given, the algorithm runs in linear time and uses linear space.*

3.3.3. Minimum-Weight Augmentation

We now generalize the algorithm from the previous section to the case where every potentially new edge e is associated with a positive cost $c(e)$. We seek a minimum-cost augmentation of a given plane graph G such that G becomes bridge-connected (while remaining plane). For a set of edges E' we define the cost of E' as $c(E') = \sum_{e \in E'} c(e)$. Given a connected convex geometric graph with n vertices, we can solve the problem in $O(n^2)$ time.

The basic idea is again to use a decomposition into (near-)cycles. The main difference from the previous problem is that in a near-cycle it is not always the best solution to add the unique edge that completes the cycle. Consider the graph given by the solid edges in Figure 3.12. The costs of the vertex pairs that are connected by dashed line segments are given in the table next to the drawing or will be specified later; all other non-adjacent vertex pairs have a very high cost. We first focus on the component to the right of and including the bold split edge e. Adding the unique edge f that completes the cycle would incur a cost of 10, whereas adding the edge set $E_1 = \{b, d\}$ would incur a cost of only 9. Adding the edge set $E_2 = \{a, d\}$ would be even cheaper, namely 7. This solution, however, has the disadvantage that e does not lie on a cycle in the component to the right of e; hence e is forced to lie on a cycle in the component to the left of e. Which option yields the better solution globally depends on the costs of edges x and y. If $c(y) - c(x)$ is greater than $c(E_1) - c(E_2)$, the optimal global solution is $E_1 \cup \{x\}$, otherwise $E_2 \cup \{y\}$ is optimal. Hence, we cannot make the decision between E_1 and E_2 in advance. Instead, we store both costs for the component to the right of e, the cost $w^+(e)$ of a cheapest augmentation that puts e on a cycle and the cost $w^-(e)$ of a cheapest augmentation that does not necessarily put e on a cycle. Note that $w^+(s) \geq w^-(s)$ for any split edge s.

Initially, we set $w^+(e) = \infty$ and $w^-(e) = 0$ for each outer edge e of G. We then compute the decomposition tree $\mathcal{T}$ of G and process its components starting from the leaves as in the algorithm described in the previous section. Other than there, we need to use dynamic

programming to find a global minimum-cost solution. Let C be a component and let $e = uv$ be the edge that is shared by C and its parent in $\mathcal{T}$. We assume that, for all edges e' of C that are distinct from e, we already have computed $w^+(e')$ and $w^-(e')$. For any set E' of vertex pairs of C, we denote the set of bridges of $C + E'$ by $\mathrm{br}_C(E')$, and we define the cost of E' as

$$\mathrm{cost}(E') = \sum_{e' \in \mathrm{br}_C(E') \setminus \{e\}} (w^+(e') - w^-(e')) + \sum_{e' \in E'} c(e').$$

The first term of the cost function describes the increase of augmentation cost stemming from the fact that e is not on a cycle in $C + E'$ and hence must be part of a cycle in a previously processed component. The second term is the cost for the edges in E'. We set

$$w^-(e) = \min_{E'} \mathrm{cost}(E')$$

and

$$w^+(e) = \min_{E',\, e \notin \mathrm{br}_C(E')} \mathrm{cost}(E').$$

We now show how to compute these values efficiently. If C is a cycle, then $w^-(e) = w^+(e) = \mathrm{cost}(\emptyset) = 0$. If C is a near-cycle, we can reduce the computation of $w^-(e)$ and $w^+(e)$ to a shortest-path problem as follows.

We say that an augmentation E' of C is *(inclusion) minimal* if, for any proper subset $E'' \subset E'$, we have that $\mathrm{br}_C(E')$ is a proper subset of $\mathrm{br}_C(E'')$, that is, any smaller set also covers fewer edges. The following lemma shows that the minimal plane augmentations of C essentially have path structure.

Lemma 3.1. *Let E' be a minimal plane augmentation of C, and let $u_1, \ldots, u_k$ be the vertices of C as they occur along C. Then $E' \cup \mathrm{br}_C(E')$ forms a path P from u_1 to u_k. The subset of vertices that is visited by P occur along P in the same order as in C.*

Proof. We first show that all vertices of P have degree at most 2, except for u_1 and u_k, which have degree at most 1. Suppose that a vertex u_i of P is incident to two distinct vertices $u_j, u_{j'}$ with $i < j < j'$. Then the edge u_iu_j cannot be a bridge, since $u_i, u_j, \ldots, u_{j'}$ forms a cycle containing this edge. Therefore also u_iu_j belongs to E' and since $j' > j$ we have $\mathrm{br}_C(E' \setminus \{u_iu_j\}) = \mathrm{br}_C(E')$. Analogously, u_i can have at most one neighbor u_j with $j < i$ in $E' \cup \mathrm{br}_C(E')$.

Next, we show that P connects u_1 and u_k. Note that u_1 is not a singleton, as it either is incident to an edge of E' or u_1u_2 is in $\mathrm{br}_C(E')$. Let u_i the vertex with the largest index that belongs to the connected component of u_1 in P. Note that $i > 0$ holds by the previous observation and suppose that $i < k$. The choice of u_i implies that u_i is not adjacent to any vertex u_j with $j > i$ in P. In particular, $u_iu_{i+1} \notin \mathrm{br}_C(E')$, which implies the existence of an edge $u_{i'}u_j$ with $i' < i$ and $j > i$. Since u_j is not in the same connected component as u_1 (this would contradict the choice of u_i we have that $u_{i'}$ also belongs to another connected component. Hence the path from u_i to u_1 must contain an edge $u_ru_{r'}$ with $1 < r < i'$ and $i' < r \leq i < j$. Such an edge would however cross the edge $u_{i'}u_j$, contradicting the planarity of E'. Hence we have $i = k$ and P connects u_1 and u_k as claimed.

It remains to show that P is connected. Suppose that P contains a vertex u_i that is not connected to u_1. Then there exists an edge $u_ju_{j'}$ with $j < i < i'$ in the path from u_1 to u_k. Hence, the edges u_ru_{r+1} with $j \leq r < i'$ are not bridges and for all other edges $u_ru_{r'} \in E'$

with $i < r < r' < i'$ we have $\mathrm{br}_c(E') = \mathrm{br}_C(E' \setminus \{u_r u_{r'}\})$, which would contradict the minimality of E'. Hence such a vertex u_i does not exist and P is connected.

The planarity of E' also implies that P contains the vertices as ordered along C. □

Lemma 3.1 shows that we can compute $w^+(e)$ by finding a shortest u_1–u_k path in the directed, weighted graph $\vec{C} = (V_C, \vec{E}_C; \ell)$ with vertex set $V_C = \{u_1, \ldots, u_k\}$, edge set $\vec{E}_C = \{u_i u_j \mid 1 \leq i < j \leq k, u_i u_j \neq e\}$, and weight

$$\ell(u_i u_j) = \begin{cases} w^+(u_i u_j) - w^-(u_i u_j), & j = i+1 \\ c(u_i u_j), & j > i+1 \end{cases}$$

for each edge $u_i u_j$ in $\vec{E}_C$. Analogously, we can compute $w^-(e)$ by adding e to $\vec{C}$ with a weight of 0. Since $\vec{C}$ is a directed acyclic graph, a shortest path can be computed in time $O(|\vec{C}|) = O(|C|^2)$ [CLRS01]. This yields an overall running time of $O(n^2)$.

We have proved the following theorem.

Theorem 3.5. *Let G be a connected convex geometric graph. Then we can find an edge set E' of minimum total weight such that $G + E'$ is bridge-connected in $O(n^2)$ time.*

3.4. Path Augmentation

In this section we consider the following two problems.

Planar k-path augmentation (k-PathAug):

> Given a planar graph G, two vertices s and t of G, and an integer $k > 1$, find a smallest set E' of vertex pairs sucht that $G + E'$ is planar and contains k edge-disjoint s–t paths.

Plane geometric k-path augmentation (geometric k-PathAug) is defined as above with "planar" replaced by "plane geometric". Note that for $k = 2$ the geometric case (geometric 2-PathAug) is a relaxed version of PECA. Both problems have a variant where the aim is to find vertex-disjoint paths. We refer to this as the *vertex variant* of (geometric) k-PathAug.

In the following we give a polynomial-time algorithm for planar 2-PathAug. We then turn to the geometric version of the problem. We show that in the worst case $n/2$ edges are needed for geometric 2-PathAug. For $k > 2$ geometric k-PathAug does not always have a solution. We give necessary and sufficient conditions for geometric 3-PathAug. We do not consider the non-geometric variant 3-PathAug, because every planar graph with at least four vertices can be triangulated and hence, can be augmented to contain three vertex-disjoint paths between any two vertices [KN05].

3.4.1. Planar 2-Path Augmentation

Theorem 3.6. *2-PathAug and its vertex variant can be solved in linear time.*

Proof. We only consider the edge variant; the vertex variant can be solved analogously. Let $G = (V, E)$ be a planar graph, let s and t be two vertices of G, and let $C_1, \ldots, C_r$ be the 2-edge connected components of G. We first consider a special case of the problem.

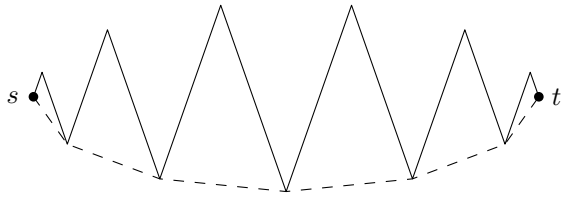


Figure 3.13.: Zig-zag path of n vertices that needs $n/2$ edges (dashed) to augment [AGH+08].

We assume that the 2-edge connected components of G form a path $C_1, \ldots, C_r$ and that $s \in C_1$ and $t \in C_r$.

For each component C_j with $2 \leq j \leq r-1$ consider the two vertices u_j and v_j of C_j that are incident to bridges. We say that C_j is a *pearl*, if $C_j + u_j v_j$ is planar. This is the case if and only if C_j has an embedding such that u_j and v_j lie on the outer face. If C_j is not a pearl, we say that C_j is a *ring*.

Let $i < k$ and let w_i and w_k be vertices of C_i and C_k, respectively, such that $G + w_i w_k$ is planar. Now assume there is a component C_j with $i < j < k$ that is a ring. Then the graph that results from contracting $C_1 \cup \cdots \cup C_{j-1}$ to u_j and $C_{j+1} \cup \cdots \cup C_r$ to v_j is $C_j + u_j v_j$. Contractions do not violate planarity, thus $C_j + u_j v_j$ is planar. This, however, violates the assumption that C_j is a ring. Hence no edge in a planar augmentation of G can "bypass" a ring. In other words, an optimal augmentation contains an edge between C_1 and the first ring, between the first and the second ring, etc., and between the last ring and C_r. If there are no rings, the optimal augmentation consists of an edge connecting C_1 and C_r, for example, between the corresponding cut vertices.

Now we consider the general case, that is, the 2-edge connected components form a tree $\mathcal{T}$. In $\mathcal{T}$, the components that contain s and t are connected by a path. This is the special case we have treated above. Obviously, any planar augmentation of the subgraph induced by the components on the path is also a planar augmentation of G. Since no ring on the path can be bypassed, there is no planar augmentation of G that uses fewer edges.

The tree of the 2-edge connected components can be computed in linear time. Finding the ring components on the path between s and t also takes linear time. Hence the whole algorithm runs in linear time. □

3.4.2. Geometric 2-Path Augmentation

Although geometric 2-PathAug appears to be a simplification of geometric PECA, it is not obvious how to take advantage of this. Therefore we consider the worst-case problem: how many edges are needed for geometric 2-PathAug in the worst case. For a zig-zag path with end vertices s and t whose vertices are in convex position $n/2$ edges are needed in order to establish two edge-disjoint s–t paths, see Figure 3.13. Abellanas et al. [AGH+08] came up with this example to show that, for trees, geometric PECA sometimes requires $n/2$ edges. They conjectured that $n/2$ edges always suffice to augment a tree to bridge-connectivity. Recently, Tóth [Tót08] confirmed this. This shows that $n/2$ edges always suffice for geometric 2-PathAug in *trees*.

We show that *any* plane geometric graph has, for any two vertices s and t, an s–t 2-path augmentation with at most $n/2$ edges. We also give a simple algorithm that finds such an augmentation in linear time. We use the fact that every geometric graph $G = (S, E)$ has a *geometric triangulation*, that is, there is a graph $T = (S, E')$ with $E \subseteq E'$ such that all faces of T except perhaps the outer face are triangles. This follows from the fact that

every simple polygon has a triangulation [dBCvKO08].

Lemma 3.2. *Let S be a finite set of points in the plane, and let $s, t \in S$. Let $G = (S, E)$ and $G' = (S, E')$ be connected plane geometric graphs such that $E \subseteq E'$. If G' contains a path of length L between s and t, then there exists an s–t 2-path augmentation of G with at most L edges.*

Proof. We can assume that G' is a triangulation of S since this does not increase the length of a shortest s–t path in G'.

Let π be a path of length L between s and t in G'. We denote its vertices by $s = v_0, \ldots, v_L = t$. We use induction on L to show that we can augment G with L edges. We start with the case $L = 1$, that is, G' contains the edge $e = \{s, t\}$. If s and t lie in the same 2-edge connected component of G, G already contains two edge-disjoint s–t paths and we're done. Otherwise we consider two cases.

If e is not in G, then s and t lie in the same 2-edge connected component of $G + e$ since G is connected. If e is already in G then e is a bridge (otherwise s and t would be in the same 2-edge connected component). Removing e from G yields two connected subgraphs G_1 and G_2 of G. Since G' is a triangulation of S that contains all edges of G, there exists an edge $e' = vw$ in G' with $v \in G_1$ and $w \in G_2$ such that e' is different from e and $G + e'$ is plane. In $G + e'$ the vertices s and t lie in the same 2-edge connected components.

We now consider $L > 1$. Given a path π of length L we first apply the induction hypothesis to the path $\pi' = v_0, \ldots, v_{L-1}$. Then we use the same argument as above to show that it suffices to add at most one edge to G to make sure that v_{L-1} and v_L are in the same 2-edge connected component. The augmented graph is plane since it is a subgraph of G'. □

For the main result of this section it remains to show that triangulations have small diameter. We need the following notation. Given a triangulation T and vertices s and t in T, we denote by $d(s, t)$ the length of a shortest s–t path in T. For a vertex v of T we denote by $N^i(v) = \{u \in T \mid d(v, u) \leq i\}$ the set of vertices of T at distance at most i from v and by $\partial N^i(v) = \{u \in T \mid d(v, u) = i\}$ the set of all vertices at distance exactly i from v. Note that $N^{i+1}(v) = N^i(v) \cup \partial N^{i+1}(v)$ for any vertex v in T and any integer $i \geq 0$.

Lemma 3.3. *Let S be a set of n points in the plane, and let $T = (S, E)$ be a triangulation of S. Then T contains a path of length at most $n/2$ between any pair of points in S.*

Proof. We first show that for any vertex v of T and for any integer $i \geq 0$ it holds that $|N^i(v)| \geq 2i + 1$ or $N^i(v) = S$. For $i = 0$ the statement clearly holds. For $i > 0$, we show that either $|\partial N^i(v)| \geq 2$ or $N^i(v) = S$. Clearly, $\partial N^i(v) = \emptyset$ implies $N^i(v) = S$ since T is connected. If $\partial N^i(v) = \{x\}$ and $N^i(v) \neq S$ then there exists a vertex y in $S \setminus N^i(v)$. In this case, however, the fact that every path from s to y must contain x implies that x is a cut vertex, which contradicts the fact that T is a triangulation, and thus biconnected. This proves our lower bound on $|N^i(v)|$.

Now consider any pair of vertices s and t in S and set $k = \lfloor n/2 \rfloor$. By the previous inequality we have that $|N^k(s)| \geq 2 \lfloor n/2 \rfloor + 1 \geq n$ and hence $N^k(s) = S$. Hence, t lies in $N^k(s)$ and, by the definition of $N^k(s)$, there exists a path of length at most $n/2$ from s to t. □

Together, Lemmas 3.2 and 3.3 yield the following theorem.

Theorem 3.7. *Let S be a set of n points in the plane, let $G = (S, E)$ be a plane geometric graph, and let s and t be two vertices of G. Then there is an s–t 2-path augmentation of G that uses at most $n/2$ edges.*

We now improve this bound for the case that the convex hull $\mathrm{CH}(S)$ of S does not contain too many points. The basic idea is to simultaneously grow neighborhoods around s and t; once $N^i(s)$ and $N^i(t)$ both contain vertices of $\mathrm{CH}(S)$ for some $i \geq 0$, there is a relatively short path connecting them.

Lemma 3.4. *Given a set S of n points in the plane, a geometric triangulation of S has diameter at most $2(n+3)/5 + h/2$, where $h = |\mathrm{CH}(S)|$.*

Proof. Let T be a triangulation of S and let v be a vertex of T. We claim the following. If $N^i(v) \cap \mathrm{CH}(S) = \emptyset$ then $|N^i(v)| \geq 3i + 1$.

We show this by induction on i. Clearly, the claim holds for $i = 0$. Now let $i \geq 1$. We apply the induction hypothesis to $N^{i-1}(v)$ and show that $|\partial N^i(v)| \geq 3$ if $N^i(v) \cap \mathrm{CH}(S) = \emptyset$. Assume, for the sake of contradiction, that $|\partial N^i(v)| \leq 2$. That means that all paths going from $N^{i-1}(v)$ to $S \setminus N^i(v)$ must pass through one of the two vertices in $\partial N^i(v)$. Hence, $\partial N^i(v)$ is a separator of cardinality 2. Let T' be the plane graph that results from T by triangulating the outer face of T (using non-straight-line edges). Since all edges in $T' - T$ connect points on the convex hull of S, which is disjoint from $N^i(v)$, it holds that $\partial N^i(v)$ is a separator of cardinality 2 of T'. This is a contradiction to the fact that every fully triangulated graph is 3-connected. Hence, our assumption is wrong, and the case $|\partial N^i(v)| \leq 2$ is ruled out. In other words, $|\partial N^i(v)| \geq 3$ for all $i \geq 1$ with $N^i(v) \cap \mathrm{CH}(S) = \emptyset$. This proves our claim.

Now let

$$k = \min\{i \mid N^i(s) \cap N^i(t) \neq \emptyset \text{ or both } N^i(s) \cap \mathrm{CH}(S) \neq \emptyset \text{ and } N^i(t) \cap \mathrm{CH}(S) \neq \emptyset\}.$$

be the first iteration where the iterated neighborhoods either meet or both have reached the convex hull of S.

Clearly there exists a path of length $2k + h/2$ between s and t. The neighborhoods give a path from s to the convex hull and from t to the convex hull and any two points on the convex hull are connected by a path of length at most $h/2$.

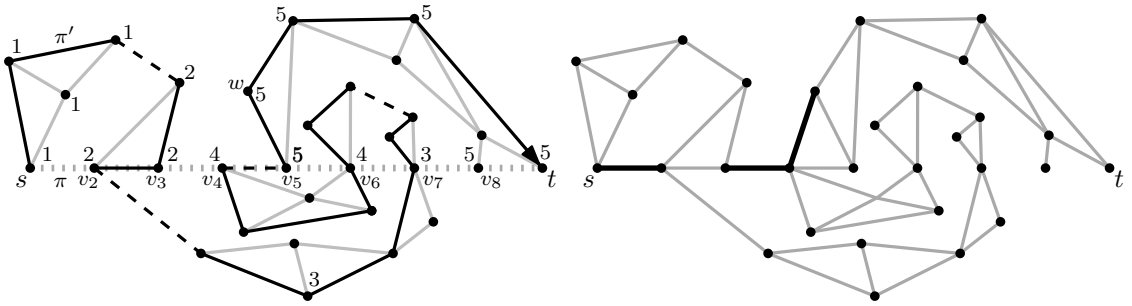
We now bound k in a similar fashion as before. We have $|N^{k-1}(s) \cup N^{k-1}(t)| \geq 5(k-1) + 2 = 5k - 3$ since the neighborhoods of s and t grow by at least two vertices as shown in the proof of Lemma 3.3 and one of them grows by at least three vertices by the claim above. On the other hand $|N^{k-1}(s) \cup N^{k-1}(t)| \leq n$. From this we get $k \leq (n+3)/5$.

Hence there exists a path from s to t with length at most $2k + h/2 \leq 2(n+3)/5 + h/2$. □

Note that the bound in Lemma 3.4 is strictly better than the bound in Lemma 3.3 if $h < (n - 12)/5$.

We now turn to the corresponding algorithmic problem. In the remainder of this subsection we show how to compute a solution to geometric 2-PathAug of size at most $n/2$ in linear time. Given a graph G, our algorithm consists of the following three steps.

1. Find any triangulation T of G.
2. Compute a shortest path π from s to t in T.
3. Construct an s–t 2-augmentation from π (whose existence follows from Lemma 3.2).



(a) A geometric graph G (solid gray, solid black and dashed black edges) with an s–t path π' (black; bridges are dashed). The s–t path π (dotted gray straight line) belongs to the triangulation T, which is not shown.

(b) The same graph G (gray edges) and the augmentation (black edges) computed by our algorithm.

Figure 3.14.: Example for the linear-time 2-path-augmentation algorithm.

Concerning step 1, note that the boundary of the outer face of a plane geometric graph is, in general, not a simple polygon. It is however *weakly simple* in the sense that segments that have a common point in the interior are actually the same segment. Algorithmically, weakly simple polygons can be handled just like simple polygons [AGH+08].

Therefore, we can apply Melkman's linear-time algorithm [Mel87] for computing the convex hull of a polygonal chain to compute the convex hull of our geometric graph. We then add edges between neighboring points on the convex hull. Now all interior faces of our graph are weakly simple polygons and can hence be triangulated in linear time [Cha91]. Let T be the resulting triangulation of G.

Concerning step 2, a shortest s–t path π in T can be found in linear time using breadth-first search.

Finally, concerning step 3, we want to show how to find an augmentation of G in linear time. We first compute a data structure that allows us to measure how we proceed along the path π by adding edges. Let π' be a simple s–t path in G. We remove all bridges of G that are used by π'. We call the resulting graph G'. We number the connected components of G' in the order in which they occur along π' and label the vertices of each component accordingly, starting with 1. Note that, by construction, all vertices on π' that have the same label lie in the same 2-edge connected component (with respect to the graph G). We denote the label of a vertex u by $\ell(u)$. See Figure 3.14a for an example.

As in the proof of Lemma 3.2, we go through the edges of π in order, starting from the edge leaving s. We consider the edges of π directed towards t. Initially, we set $j = 1$. Throughout the algorithm we maintain the following two invariants.

(I1) It holds that $j \geq \ell(v)$ for each vertex v of π whose incoming edge has already been processed.

(I2) All vertices of π' with label up to j lie in the same 2-edge connected component of G.

Both invariants clearly hold for $j = 1$. Together, the invariants yield the correctness of the algorithm since π ends in t and hence, according to invariant (I1), we have $j = \ell(t)$ after the last step and thus, by invariant (I2), s and t belong to the same 2-edge connected component.

We now describe the algorithm and show that it preserves the two invariants. We distinguish two main cases based on the values of j and of the label $\ell(v)$ of the endpoint of the current edge $e = uv$ of π.

If $\ell(v) \leq j$ (as for $e = v_5v_6$ in Fig. 3.14a), we simply advance to the next edge of π. Clearly, this preserves both invariants.

If $\ell(v) > j$, we add a suitable edge to G as follows. If e is not in G (as for $e = v_3v_4$ in Fig. 3.14a), we simply add e to G and set j to $\ell(v)$. Otherwise (as for $e = v_4v_5$ in Fig. 3.14a), e is a bridge of G lying on the path π' and we have $\ell(v) = \ell(u) + 1$. In the triangulation T, the edge $e = uv$ bounds at least one triangle. Let uwv be such a triangle. Hence, there are two possibilities for adding an edge to G; either uw or wv. If $\ell(w) > \ell(u)$ (as for $u = v_4$ and $v = v_5$ in Fig. 3.14a), we add the edge uw (edge v_4w in Fig. 3.14a) and set j to $\ell(w)$ (to 5 in Fig. 3.14a). Otherwise, we add the edge wv and set j to $\ell(v)$.

Clearly, the number of edges we add is bounded by the length of the path π. See Fig. 3.14b for the edges that are added in the case of the graph from Fig. 3.14a.

We now argue that the algorithm preserves our invariants. By our choice of j, it is clear that invariant (I1) holds. To prove (I2), let a and b be the two endpoints of the newly added edge. Let a' be a vertex of π' closest to a in G' (in terms of graph distance) and let π_a be a shortest a–a' path. Let b' and π_b be defined analogously. Since π_a and π_b lie in G', $\ell(a) = \ell(a')$ and $\ell(b) = \ell(b')$. As $\ell(a) \neq \ell(b)$, π_a and π_b are disjoint; they "live" in different 2-edge-connected components of G. The newly added edge ab together with π_a and π_b and the subpath of π' that connects a' and b' form a simple cycle. This shows that, after adding the new edge ab, vertices a' and b' belong to the same 2-edge connected component of G. By invariant (I2) we have that a' lies in the same 2-edge connected component as s. Now we use transitivity and the fact that, after the addition of ab, variable j is set to $\ell(b) = \ell(b')$. This yields that indeed, after adding ab, all vertices of π' with label at most j lie in the same 2-edge connected component. In summary, we have proved the following theorem.

Theorem 3.8. *Let S be a set of n points in the plane, let $G = (S, E)$ be a plane geometric graph, and let s and t be vertices of G. Then there exists a set E' of at most $n/2$ vertex pairs such that $G + E'$ is a plane geometric graph that contains two edge-disjoint s–t paths. Such a set of vertex pairs can be computed in $O(n)$ time.*

3.4.3. Geometric 3-Path Augmentation

In this section we consider the problem of augmenting geometric graphs to contain more than two disjoint s–t paths while staying plane. The planar case obviously always has a solution, because every planar graph can be triangulated and a planar triangulation is always 3-connected. Hence we focus on the plane geometric cases in this section. In the following we give necessary and sufficient conditions for when plane geometric s–t 3-augmentation has a solution.

We first consider the vertex version of the problem, that is, given a geometric graph $G = (S, E)$ and two vertices s and t of G, add edges to G such that G contains three vertex-disjoint s–t paths.

The Vertex-Disjoint Case

Let $T = (S, E)$ be any plane geometric triangulation, and let s and t be any two vertices of T. An edge between two vertices of the convex hull that does not belong to the convex

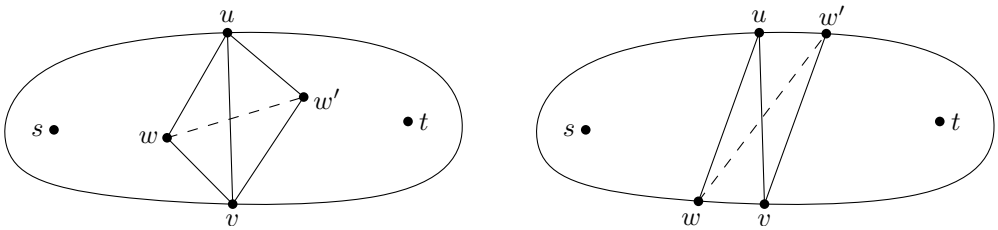


Figure 3.15.: Removing an s–t separating chord uv by flipping (a). If the new edge ww' is again an s–t separating chord, then uw or vw is an s–t separating chord that is closer to s than uv (b).

hull itself is called a *chord*. A chord $e = \{u, v\}$ is *s–t separating* if s and t lie in different connected components of $T \setminus \{u, v\}$.

Obviously there exist three vertex-disjoint s–t paths in T if and only if T does not contain an s–t separating chord. Hence we can rephrase our original question in the following form: Let G be any plane geometric graph. Can we triangulate G such that the resulting triangulation T_G contains no s–t separating chord? The following theorem states that this question can be answered in the affirmative.

Theorem 3.9. *Let S be a finite set of points in the plane, let $G = (S, E)$ be a plane geometric graph, and let s and t be any two vertices of G. If G contains no s–t separating chord, we can compute a triangulation T_G that contains three vertex-disjoint s–t paths.*

Proof. In the first step we add all edges of the convex hull to G and compute any triangulation of the interior. We can give a total ordering to the s–t separating chords of the triangulation by their facial distance from s. Let uv be the chord that is closest to s. Let uvw be the triangle that is on the same side as s with respect to uv and let uvw' be the other triangle bounded by uv. As u and v lie on the convex hull, we can *flip* the chord uv, that is, replace uv by the edge ww' without destroying planarity; see Figure 3.15a. If the new edge ww' was an s–t separating chord, then one of the edges uw and vw would have to be an s–t separating chord that is closer to s than uv, contradicting the choice of uv; see Figure 3.15b. Hence we have removed an s–t separating chord without introducing a new one. Inductively we obtain the desired triangulation T_G. □

The Edge-Disjoint Case

In this section we consider the problem of adding edges to a given plane geometric graph G such that for two fixed vertices s and t of G there exist three edge disjoint s–t paths. Since every plane geometric graph can be triangulated, we characterize the triangulations that contain three edge-disjoint s–t paths.

Theorem 3.10. *Let S be a finite set of points in the plane, let $T = (S, E)$ be a geometric triangulation of S, and let s and t be any two vertices of T. Then T contains three edge-disjoint s–t paths if and only if s and t have degree at least 3.*

Proof. Clearly the degree conditions are necessary for the existence of three edge-disjoint paths in T. We show that they are also sufficient. We use Menger's Theorem, which says that a graph is k-connected if and only if it contains k vertex-disjoint paths between any

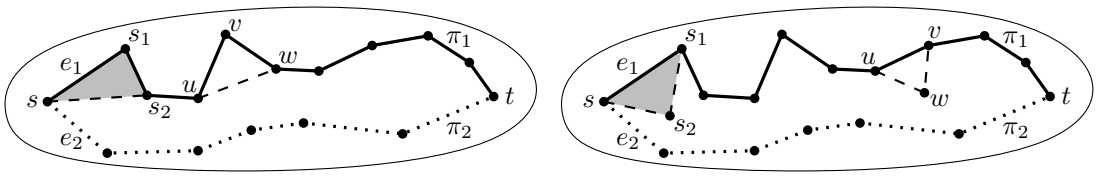


Figure 3.16.: Hand rail construction along path π_1.

pair of vertices. Hence, since T is biconnected, there exist two vertex-disjoint s–t paths π_1 and π_2. We now show that we can find a third s–t path π_3 that is edge-disjoint from π_1 and π_2.

We start our construction of π_3 by constructing a path to a vertex s^* on $(\pi_1 \cup \pi_2) \setminus \{s\}$. Let e_1 and e_2 be the first edges of π_1 and π_2, respectively. Since s has degree at least 3 and T is triangulated, there exists a triangle incident to s whose boundary contains exactly one of the edges e_1 and e_2. Let s, s_1, and s_2 be the vertices of this triangle. We assume without loss of generality that $ss_1 = e_1$. We start π_3 with the edge ss_2, which neither belongs to π_1 nor to π_2. If s_2 lies on π_1 (see Figure 3.16a), we let $s^* = s_2$. Otherwise (see Figure 3.16b) we append the edge s_2s_1 to π_3 and let $s^* = s_1$. Note that s_2s_1 neither belongs to π_1 nor to π_2.

We now show that given the vertex s^* on π_1, we can continue the construction of π_3 by using π_1 as a "hand rail". Let u be a vertex on π_1 (initially $u = s^*$), and let v be the vertex next to u on π_1 in the direction of t. Then the edge uv bounds a triangle on at least one side. If $v = t$ then, due to $\deg(t) \geq 3$, there exists a triangle whose boundary contains ut and two other edges that do not belong to π_1 and π_2. Hence, we can use these to connect π_3 to t. If $v \neq t$, we consider the triangle $\{u, v, w\}$ whose boundary contains uv. If w is incident to v on π_1 in the direction of t, then we append the edge uw to π_3, see Figure 3.16a. Otherwise we append edges uw and wv to π_3, see Figure 3.16b. In neither of the two cases do we use edges that belong to π_1 or π_2.

Note that the path we construct in this way is not necessarily simple. This can be corrected by removing cycles. □

3.5. Concluding Remarks

We have studied the complexity of several connectivity augmentation problems. We have showed that PECA, PVCA, and their geometric variants are all NP-hard. On the positive side, we have given efficient algorithms for 2-PathAug and its vertex variant on planar graphs. Further, we have studied worst-case bounds for geometric 2-PathAug, and we have fully characterized geometric graphs that can be augmented to contain three (vertex-)disjoint s–t paths.

In particular, we have seen that although the problem of bridge- or biconnecting a given connected graph is polynomial-time solvable, adding the constraint of preserving planarity makes these problems much more difficult and shows a different face of planarity than the helpful one. On the other hand, planarity was still helpful in many cases, in particular the strong planarity conditions of convex geometric graphs allow to solve PECA for convex

geometric graphs, even in the much more difficult weighted case. This is a stark contrast to the result of Frederickson and Ja'Ja [FJ81] that finding a minimum-weight edge set that biconnects a given tree is NP-hard, even if the weights are restricted to the set $\{1, 2\}$.

Open Problems. We conclude with some open questions. The most important open questions are whether geometric PECA and geometric PVCA admit efficient constant-factor approximations. This would nicely complement the negative results on these problems. A worthwhile direction for future work may also be to study the worst-case behavior of augmenting a $(c-1)$-connected plane geometric graph to be c-connected for $c > 2$.

More questions arise from a look at Table 3.1; what are the correct entries for the remaining open cells? Is it possible to adapt Kant's optimal augmentation algorithms for outer-planar graphs [Kan96] to the weighted case?

Finally, what is the complexity of the path augmentation problems for which we have given worst-case bounds in this chapter? In particular, can geometric 2-PathAug be solved efficiently?

Chapter 4

Switch Graphs

Switch graphs offer enhanced modeling capabilities over ordinary graphs. A switch consists of a set of edges that share a common vertex. A configuration of a switch graph picks one edge from each switch. Therefore, a switch graph describes a family of graphs and a configuration describes a concrete member of this family. By construction, switch graphs are well-suited for modeling graph-theoretic problems that involve structural decisions. Given a switch graph, we wish to find out whether the corresponding family of graphs contains a member that satisfies a specified graph property.

We derive a variety of results on the algorithmics of switch graphs. On the negative side we prove hardness of the following problems: Given a switch graph, does it possess a bipartite/planar/triangle-free/Eulerian configuration? On the positive side we design fast algorithms for several connectivity problems in undirected switch graphs, and for recognizing acyclic configurations in directed switch graphs.

The chapter is based on joint work with Bastian Katz and Gerhard Woeginger [KRW10].

4.1. Introduction

What is a switch graph? A *switch* s on an underlying vertex set V is a pair (p_s, T_s) where $p_s \in V$ is the *pivot* vertex and where $T_s \subseteq V$ is a non-empty set of *target* vertices. The vertex set V and some set S of switches on V together form a *switch graph* $G = (V, S)$. A *configuration* of a switch graph is a mapping $c : S \to V$ such that $c(s) \in T_s$ for all $s \in S$. The configuration selects exactly one edge $e_c(s) := \{p_s, c(s)\}$ for every switch $s \in S$, and thus yields a corresponding multi-set $E_c = \{e_c(s) : s \in S\}$ of edges. The corresponding multi-graph is denoted $G_c = (V, E_c)$; see Figure 4.1 for an illustration. Biologically speaking, a switch graph represents the genotype of an entire population of graphs, and every configuration specifies the phenotype of one concrete member in this population.

A brief history of switch graphs. Over the last 30 years a huge number of fairly unrelated combinatorial structures has been introduced under the name *switch graph* or *switching graph*; see the introduction of the paper by Groote and Ploeger [GP08] for some pointers to the literature. The switching graph model of Meinel [Mei89] comes very close to the model that is investigated in this chapter. Another somewhat restricted type of switch graph has been introduced by Cook [Coo03] who studied cyclic configurations as an abstraction of certain features in Conway's game of life. In Cook's model the vertices

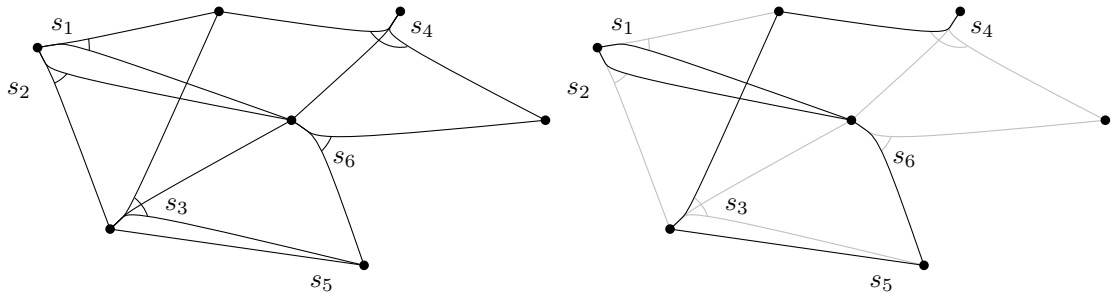


Figure 4.1.: To the left: A switch graph G with six switches, where s_5 has only a single target. To the right: A configuration yielding a multigraph G_c. Edges belonging to the same switch are marked by an arc at their pivot. The edges picked by the configuration are shown in black, the grey edges are not picked.

are not allowed to have degrees higher than three, and every switch has an obligatory incident edge that must show up in every configuration. Reinhardt [Rei09] essentially studies Cook's model, but drops the constant degree constraint. Reinhardt constructs a polynomial-time $O(|V|^4)$ algorithm that decides whether there exists a configuration that contains a simple path between two prespecified vertices. He also links switch graphs to certain matching problems in computational biology [SGYBD05]. We note that Cook's and Reinhardt's switch graph models can both easily be emulated by our switch graph model. Huckenbeck [Huc93, Huc97] studied a related but more general problem where the configuration of *valves* at nodes decides which pairs of incoming and outgoing edges of a node may be used by a path.

Groote and Ploeger [GP08] concentrate on switch graphs with *binary* switches (where every target set contains two elements). Their work is motivated by the problem of solving Boolean equation systems, which is known to be equivalent to the model checking problem in modal μ-calculus, a problem whose complexity is yet open. Previously, Groote and Keinänen showed that solving Boolean equation systems can be reduced to solving *parity games* [GK04]. A parity game is a two player game that is played on a directed graph whose vertices are labeled with natural numbers, called *priorities*. From a given starting node, the two players, usually called 0 and 1, alternatingly move to an adjacent node. This results in an infinite path. If the smallest priority that occurs infinitely often in the path is even, Player 0 wins, otherwise Player 1 wins. It is not hard to see that the player decisions at each node can be encoded as switches; at a node, a player must choose exactly one of the outgoing edges. A winning strategy for player 0 then consists of a switch setting (that is, a decision for each node), such that on all cycles the lowest occurring priority is even.

Motivated by this observation, Groote and Ploeger [GP08] study the complexity of finding configurations of given switch graphs that have certain graph properties. For instance, they show that in directed binary switch graphs, one can decide in polynomial time whether there is a configuration that connects (respectively disconnects) two prespecified vertices. This chapter was heavily inspired by the conclusions section of their work; our results in Theorems 4.3, 4.5, and 4.9 answer open questions that have been posed there [GP08].

Contribution. Let $\mathcal{P}$ be any graph property and let G be a switch graph. We say that a configuration c of G has property $\mathcal{P}$ if G_c satisfies property $\mathcal{P}$. Every graph property $\mathcal{P}$ naturally leads to a corresponding algorithmic problem on switch graphs: Given a switch graph G, does there exist a configuration with property $\mathcal{P}$? We will derive a collection of

positive and negative results for various graph properties.

- It is NP-hard to decide whether a given switch graph has a configuration that is (a) bipartite, (b) planar, or (c) triangle-free. The three hardness proofs are presented in Section 4.3.

- We establish a number of matroid properties for switch graphs that possess a connected configuration. This yields a simple $O(|S| + |V|^2)$ time greedy algorithm for finding a configuration that minimizes the number of connected components (and of course also settles the question whether there is a connected configuration); see Section 4.4.

- We provide a linear-time algorithm to detect a configuration that connects two given vertices in an undirected switch graph. This substantially improves the $O(n^4)$ time complexity of Reinhardt's result [Rei09]; see Section 4.5.

- Finding a configuration in which all vertex degrees are even is easy, but finding a configuration with a Eulerian cycle is NP-hard for forward directed switch graphs as well as for undirected switch graphs. Moreover, it is NP-hard to find a configuration that is biconnected (for undirected switch graphs) or strongly connected (for forward directed switch graphs); see Section 4.6.

- Deciding whether a forward directed switch graph allows an acyclic configuration can be done in linear time. In contrast to this, finding a configuration that minimizes the number of directed cycles is NP-hard; see Section 4.7.

We stress that our negative results hold in the most restricted binary switch model, whereas our positive results apply to the general model.

4.2. Basic Definitions

Let $G = (V, S)$ be a switch graph. For a subset $S' \subseteq S$ of switches and a configuration c, we denote $E_c(S') = \{e_c(s) : s \in S'\}$ and $G_c(S') = (V, E_c(S'))$. We denote $V(s) := T_s \cup \{p_s\}$ and $V(S') := \bigcup_{s \in S'} V(s)$. For $S' \subseteq S$ and $V' \subseteq V$, we denote by $S'(V') := \{s \in S' \mid V(s) \subseteq V'\}$ the set of *inner switches* of V'. Observe that $V(S'(V')) \subseteq V'$. A switch graph has *fan-out* k if $|T_s| \leq k$ for all $s \in S$. It is called *binary* if $|T_s| \leq 2$ holds for all $s \in S$. Throughout this chapter we will use $n := |V|$, $m := |S|$, and $\overline{m} := \sum_s |T_s|$. Note that $m \leq \overline{m} \leq km$ for the fanout k of G.

Although this chapter mainly deals with undirected graphs, all definitions easily carry over to directed switches and directed multi-graphs. In a *forward* switch $s = (p_s, T_s)$, arcs must be directed from pivot to target. In a *reverse* switch $s = (T_s, p_s)$, arcs must be directed from target to pivot. A *directed switch graph* may contain both, forward and reverse switches. A *forward directed switch graph* contains only forward switches.

Note that all problems we consider in this chapter ask for configurations of a given switch graph with properties that can be tested in polynomial time on usual graphs. Since every switch graph has at most n^m configurations, all NP-hard problems presented in this chapter are also NP-complete.

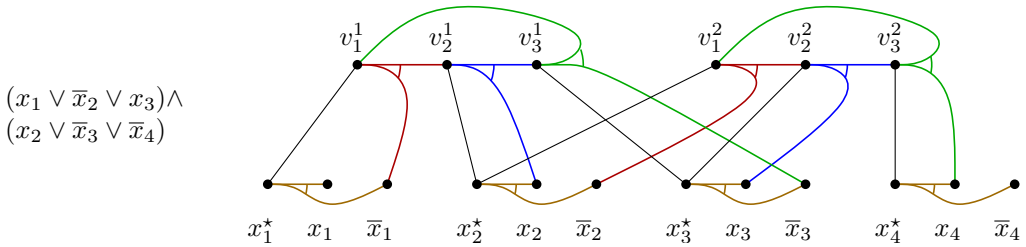


Figure 4.2.: Reduction of 3-Sat to SwitchTriangleFree

4.3. Bipartite, Planar, and Triangle-Free Graphs

In this section, we show hardness of finding configurations that are bipartite, triangle-free or planar.

Theorem 4.1. *For binary undirected switch graphs, it is NP-hard to decide if there is a bipartite configuration* (SwitchBipartite).

Proof. We sketch a reduction from SetSplitting: Given a ground set $X = \{x_1, \ldots, x_n\}$ and a set T of 3-element subsets of X, it is NP-hard to decide whether there is a partition of X into two sets X_1, X_2, such that every $t \in T$ has non-empty intersection with both, X_1 and X_2 [GJ79]. For a given instance of SetSplitting, we construct a switch graph $G = (V, S)$, containing vertices $x_1, \ldots, x_n$ for the elements of X. For each triplet $t_i \in T$ we introduce a switch $s_i = (x_j, t_i - \{x_j\})$ for an arbitrary $x_j \in t_i$.

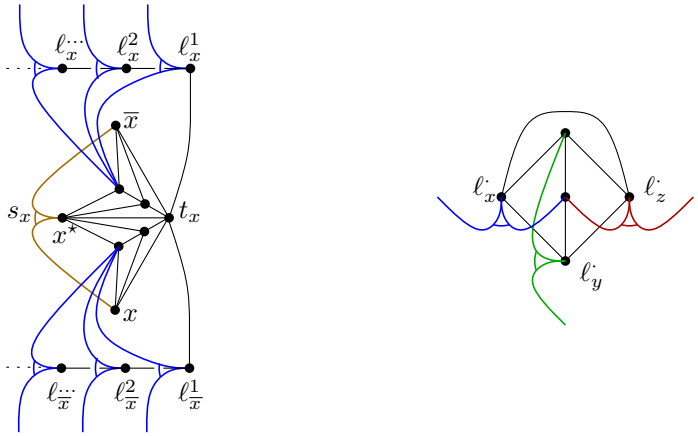
Every solution X_1, X_2 to SetSplitting yields a bipartite configuration: Color the vertices x_i according to X_1, X_2. Then every triplet t contains both colors, which allows to set the corresponding switch to connect two vertices of distinct colors. Conversely, every bipartition of some configuration G_c induces a bipartition of the x_i. For any triplet in T, the switches prevent the corresponding three vertices from receiving all the same color, and thus the induced partition yields a solution to SetSplitting. □

Theorem 4.2. *For binary undirected switch graphs, it is NP-hard to decide if there is a triangle-free configuration* (SwitchTriangleFree).

Proof. The proof is by reduction from 3-Sat. Let φ be an instance of 3-Sat. Without loss of generality we assume that each clause contains three different variables. For each variable x_i we create three vertices $x_i^\star$, x_i and $\overline{x}_i$, and a variable switch $s_i = (x_i^\star, \{x_i, \overline{x}_i\})$. For every clause C_j, we add three new vertices v_1^j, v_2^j, v_3^j. If the kth literal in clause C_j corresponds to variable x_i, we introduce an edge $(v_k^j, \{x_i^\star\})$. If it is x_i, we introduce a switch $(v_k^j, \{v_{k+1}^j, \overline{x}_i\})$. If it is $\overline{x}_i$, we introduce a switch $(v_k^j, \{v_{k+1}^j, x_i\})$, defining $v_4^j := v_1^j$. See Figure 4.2 for an example.

Intuitively, the variable switch is supposed to pick the true literal. A clause switch can only connect outside the clause, if its corresponding literal is satisfied. Consequently, in a satisfying truth assignment we can connect at least one switch of every clause to the outside, thus avoiding all triangles. Conversely, a triangle-free configuration specifies a truth assignment for the variables such that every clause contains at least one satisfied literal. Otherwise, the corresponding clause switches would induce a triangle. □

Theorem 4.3. *For binary undirected switch graphs, it is NP-hard to decide if there is a planar configuration* (SwitchPlanar).



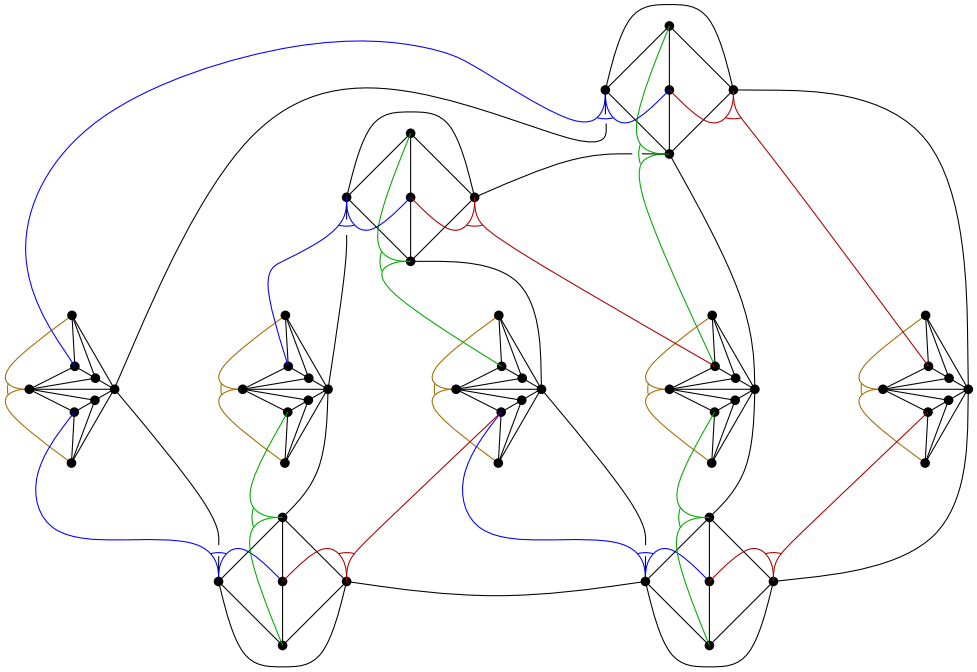
(a) (b)

Figure 4.3.: Gadgets for the reduction of Monotone Planar 3-Sat to SwitchPlanar. The variable gadget forces that either upper or all lower literal switches have a target outside the variable gadget (a). The clause gadget forms a K_5 if all its switches choose a target inside the clause (b).

Proof. The proof is by reduction from Monotone Planar 3-Sat. Planar 3-Sat is a well-known NP-hard restriction of 3-Sat where additionally the variable–clause graph is assumed to be planar. Monotone planar 3-Sat is even more restricted: the literals of each clause must be either all positive or all negative. Moreover, the variable–clause graph can be drawn in the plane without crossings such that all the variables are on the x-axis, the clauses with positive literals are above the x-axis and the clauses with negative literals are below the x-axis. Monotone planar 3-Sat is NP-hard [dBK10].

Let φ be an instance of planar monotone 3-Sat. For every variable x, we introduce a variable gadget as depicted in Figure 4.3a with one variable switch s_x and switches with pivots ℓ_x^i and $\ell_{\overline{x}}^i$ for every occurrence i of a literal x or $\overline{x}$ in a clause. For every clause C, we introduce a clause gadget as depicted in Figure 4.3b, which basically is a K_5 of which three edges can be disabled by setting a switch appropriately. We identify the pivots of these switches (and the pivots themselves) with the vertices ℓ_x^i or $\ell_{\overline{x}}^i$ for the respective literal x or $\overline{x}$ and an i induced by the drawing of φ. An example is given in Figure 4.4.

A solution to φ induces a planar drawing by switching s_x for a true x to x and the switches with pivots ℓ_x^i to the inner of the variable gadget, the switches with pivots $\ell_{\overline{x}}^i$ to the inner of the clause gadget. For a false x, we set switches accordingly. Conversely, in a planar drawing at most two switches of a clause gadget may be switched to the inner of the clause, which otherwise would be a K_5. If, in turn, some switch with pivot ℓ_x^i is switched to the inner of the variable gadget, the switch s_x must be switched to x, since otherwise contraction of the path $t_x \ell_x^1 \dots \ell_x^i$ would make the upper half of the variable gadget a K_5. Analogously, any switch with pivot $\ell_{\overline{x}}^i$ being switched to the inner of the variable clause forces s_x to be switched to $\overline{x}$. □



$$(x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_3 \vee \overline{x}_4 \vee \overline{x}_5)$$

Figure 4.4.: Example reduction of Monotone Planar 3-Sat to SwitchPlanar.

4.4. Global Connectivity

In this section we discuss the question whether a given switch graph $G = (V, S)$ has a connected configuration. It turns out that this question has many ties to matroid theory, which allows us to invoke some powerful machinery from mathematical programming.

In this context a *structure* is a tuple $(E, \mathcal{F})$ consisting of a ground set E and a set system $\mathcal{F} \subseteq 2^E$ over the ground set. Let further $c\colon E \longrightarrow \mathbb{R}$ be a cost function. Many combinatorial optimization problems can be cast into this form such that the goal is to find a set $X \in \mathcal{F}$ with maximum total weight. In its full generality this problem is very difficult. It becomes more feasible, if we pose additional requirements on the structure. A *matroid* is a structure $(E, \mathcal{F})$ that satisfies the following three properties.

(i) $\emptyset \in \mathcal{F}$;

(ii) $Y \in \mathcal{F}$ and $X \subseteq Y$ implies $X \in \mathcal{F}$;

(iii) If $X, Y \in \mathcal{F}$ with $|X| > |Y|$, then there is an $x \in X \setminus Y$ such that $Y \cup \{x\} \in \mathcal{F}$.

The sets in $\mathcal{F}$ are also called *independent* sets. For matroids, the maximization problem mentioned above can be solved by a simple greedy algorithm [KV08]. The algorithm starts with the empty set and successively either adds elements or discards them, depending on whether adding an element yields a set belonging to $\mathcal{F}$. If the elements are processed in non-increasing order of weight, the algorithm is guaranteed to find a solution of maximum weight. To employ the greedy algorithm it is necessary to have a fast *independence test* that checks for a given set $X \in \mathcal{F}$ and an element $x \in E$ whether $X \cup \{x\} \in \mathcal{F}$. The running time of the algorithm is usually dominated by n applications of such an independence test. In the following we omit the cost function since we are only interested in solutions of maximum size, instead of weight.

For the problem of connecting a given switch graph, we propose two approaches, one that immediately shows membership in $\mathcal{P}$, but results in a complicated and not very practical algorithm and a second algorithm that requires a longer argument but is both faster and simpler.

First, we consider two matroids $(E, \mathcal{I}_1)$ and $(E, \mathcal{I}_2)$ that both have the same ground set E, which is the set of all the multi-edges over V that can possibly result from any switch in S. The set system $\mathcal{I}_1$ consists of all cycle-free subsets of E. The set system $\mathcal{I}_2$ consists of all subsets of E that contain at most one multi-edge from each switch. Then $(E, \mathcal{I}_1)$ forms a *graphic matroid* and $(E, \mathcal{I}_2)$ forms a *partition matroid*. Obviously, the switch graph $G = (V, S)$ has a connected configuration, if and only if there exists a set $E' \subseteq E$ of cardinality $n - 1$ that belongs to both $\mathcal{I}_1$ and $\mathcal{I}_2$. This is a standard matroid intersection problem, which can be solved in polynomial time [Edm69].

It is not hard to see that the intersection $(E, \mathcal{I}_1 \cap \mathcal{I}_2)$ itself does not form a matroid. In the following, we will model the problem in terms of a single matroid, which yields simpler and faster algorithms. Our approach is based on a third structure $(S, \mathcal{I}_3)$ that is defined over the ground set S of switches. A subset $S' \subseteq S$ lies in $\mathcal{I}_3$, if there exists a configuration c such that $E_c(S')$ is cycle-free (or in other words, such that $E_c(S')$ belongs to $\mathcal{I}_1$).

Theorem 4.4. *The structure $(S, \mathcal{I}_3)$ forms a matroid.*

Proof. Clearly the set system $\mathcal{I}_3$ contains the empty set and is closed under taking subsets. It remains to show that for two independent sets $A, B \subseteq S$ with $|A| < |B|$, there is an $s \in B \setminus A$ such that also $A \cup \{s\}$ is independent.

Since A and B are independent, there exist configurations a and b for which the corresponding edge sets $E_a(A)$ and $E_b(B)$ are cycle-free. Among all such configurations a and b, we consider a pair that maximizes the number of switches that are in $A \cap B$ and that configure into the same edge both in configuration $E_a(A)$ and in configuration $E_b(B)$; such switches are called *good* switches. Since $E_a(A)$ and $E_b(B)$ are cycle-free, they belong to $\mathcal{I}_1$ in the underlying graphic matroid. Since $|E_a(A)| = |A| < |B| = |E_b(B)|$, there exists an edge $e \in E_b(B) \setminus E_a(A)$ such that $E_a(A) \cup \{e\}$ is cycle-free.

Let $s_e \in B$ denote the switch that in configuration b generates edge e. We claim that this switch s_e cannot be in A: Otherwise configuration a would configure this switch s_e into an edge f. Then we can modify configuration a into a new configuration c by switching s_e into e instead of f. The resulting edge set $E_c(A)$ is still cycle-free, whereas the number of good switches has increased. That is a contradiction. Hence $s_e \notin A$, and $A \cup \{s_e\}$ is an independent set of switches. □

Our next goal is to get a better understanding of independence in $(S, \mathcal{I}_3)$.

Lemma 4.1. *A set $S' \subseteq S$ is independent if and only if $|T| < |V(T)|$ holds for all $T \subseteq S'$.*

Proof. One direction of the proof is easy: If there is a $T \subseteq S'$ with $|T| \geq |V(T)|$, then every configuration c induces a cycle on T since $|E_c(T)| = |T| \geq |V(T)|$ holds.

For the other direction of the proof, we consider a set $S' \subseteq S$ that contains some switch $s \in S'$ for which $S'' = S' - \{s\}$ is independent. We will show that then either S' itself is independent, or that it contains an appropriate subset T with $|T| \geq |V(T)|$. Indeed, since S'' is independent there exists a configuration c for which $E_c(S'')$ is cycle-free. Adding an arbitrary edge e from switch s to $E_c(S'')$ produces a configuration c for S' whose edge set $E := E_c(S'') \cup \{e\}$ contains at most one cycle. If the edge set forms an acyclic graph, there is nothing to show. We hence assume that is contains a single cycle. Let C_0 denote the

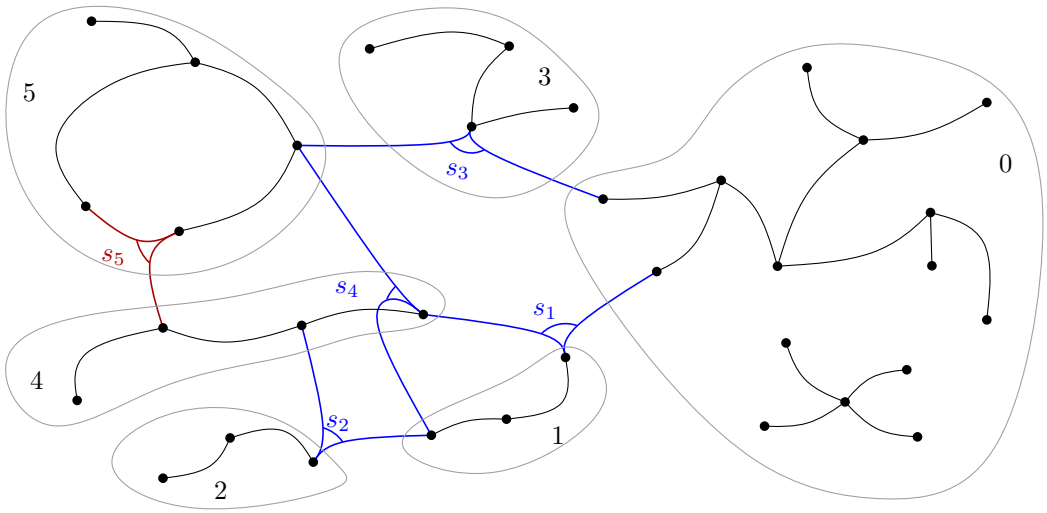


Figure 4.5.: A proper switch sequence $s_1, \ldots, s_5$ that eventually breaks the cycle. Switches are reinserted changing s_5, s_4, s_1.

connected component that contains the cycle, and let V_0 denote the set of all vertices in $V(S')$ that are not in C_0.

We work through a number of removal phases. In the ith phase ($i \geq 1$) we select a switch $s_i = (p_i, T_i)$ that contributes an edge e_i to component C_{i-1}, and whose target set T_i contains a target t_i^* in some set $V_{j(i)}$ with $0 \leq j(i) \leq i-1$. We remove the edge e_i from the edge set, and thus split component C_{i-1} into two connected parts. The part containing the cycle becomes the new component C_i, and the vertices of C_{i-1} in the other (cycle-free) part form the set V_i. Then the $(i+1)$th removal phase starts.

There are two possibilities how this process can terminate: Either (i) there is no appropriate switch with a target in $V_0, \ldots, V_{i-1}$, or (ii) removing the edge destroys the cycle in component C_{i-1}. In case (i), we choose T as the set of switches in component C_{i-1}; then $|T| \geq |V(T)|$, and we are done. In case (ii) we will show how to reinsert and how to reconfigure the removed edges and switches step by step in reverse order $s_k, s_{k-1}, \ldots, s_1$ so that the resulting edge set is cycle-free (here k denotes the number of the last phase). See Figure 4.5 for an illustration.

Throughout the procedure we will maintain the following invariant: Just after the reconfiguration of switch s_i ($1 \leq i \leq k$), there exists an index $\ell(i)$ with $0 \leq \ell(i) < i$, such that the vertex set $V_{\ell(i)} \cup \bigcup_{h \geq i} V_h$ forms a cycle-free connected component with respect to the current edge set. This component is called the *crucial* component; intuitively speaking we will make it grow until it covers all of $V(S')$. We start the growing process with switch s_k, which by definition has a target t_k^* in the set $V_{j(k)}$ with $j(k) < k$. By reinserting the edge $\{p_k, t_k^*\}$ for switch s_k and by setting $\ell(k) := j(k)$, we satisfy the invariant. In handling a switch s_i with $i < k$ we distinguish two cases: First, if $i \neq \ell(i+1)$ then we simply reinsert its old edge e_i and keep $\ell(i) := \ell(i+1)$. This merges the vertices in V_i into the crucial component while maintaining the invariant since $j(i) \leq i-1$. In the second case $i = \ell(i+1)$. We insert the new edge $\{p_i, t_i^*\}$, and set $\ell(i) := j(i)$. This merges the vertices in $V_{j(i)}$ into the crucial component, and again maintains the invariant. This reconfiguration process eventually produces a cycle-free configuration for S', and thus completes the proof. □

The statement of Lemma 4.1 is combinatorial, but its proof is algorithmical and yields as a by-product a fast independence test for the matroid $(S, \mathcal{I}_3)$: Given an acyclic configuration

of an independent set S'' we can check in $O(n + \overline{m})$ time whether a given switch s can be added to S'' without destroying independence: The independence of S'' implies that $|S'' + s| \leq n$, which also bounds the number of removal phases. To achieve selection of removable switches within a total of $O(n + \overline{m})$ time, we direct all edges in $E_c(S')$ that are not part of the cycle to point away from it. Whenever a switch s_i is removed, we use this information to mark all vertices in V_i. Obviously, this only adds $O(n + \overline{m})$ time. A switch s is a candidate if it has a marked target and both p_s and $c(s)$ are unmarked. A set of candidates can be maintained in $O(\overline{m})$ total time.

If the test is positive, we obtain a corresponding cycle-free configuration for $S'' \cup \{s\}$. If the test is negative, we get the final component C_{k-1} that contains the cycle. Let U denote the set of all switches in S'' that contribute an edge to C_{k-1}. Then $|U| = |V(C_{k-1})| - 1$, and none of the switches in U has a target outside of $V(C_{k-1})$. Hence in any cycle-free configuration of S'' the switches in U induce a connected graph on $V(C_{k-1})$; such a set U of switches is called a *tight* set. These ideas lead to the following theorem, which is the main result of this section. Note that, as a special case, the theorem yields a polynomial-time algorithm for recognizing switch graphs with connected configurations.

Theorem 4.5. *For a given switch-graph on n vertices whose underlying multigraph has $\overline{m}$ edges, we can determine in $O(n^2 + n\overline{m})$ time a configuration that minimizes the number of connected components.*

Proof. Any basis $\mathcal{B}$ of the matroid $(S, \mathcal{I}_3)$ yields a cycle-free configuration with the maximum number of edges, and hence a configuration with the minimum number of connected components; the switches not in $\mathcal{B}$ can then be set arbitrarily. Hence it is enough to determine a basis, and this is done by the standard greedy algorithm.

We start with the empty set, and test the switches one by one. If the test is positive, we add the switch and update the cycle-free configuration. If the test is negative, we discard the switch and contract all switches in the corresponding tight set U. These contractions can be done in overall $O(n\,\alpha(n))$ time by using a union-find data structure. Every test on a non-trivial graph costs $O(kn)$ time. Every positive test adds an edge to a cycle-free edge set; hence there are at most $n - 1$ of these tests. Every negative test on a non-trivial graph contracts some vertices; hence there are at most $n - 1$ of these tests. All negative test on trivial graphs (that have been contracted to a single vertex) together cost $O(\overline{m})$ time. All in all, this yields the claimed time complexity. □

Global Connectivity and Bipartite Matching. Although not obvious at first sight, there is a strong similarity between global connectivity in switch graphs and bipartite matching. Both problems can be expressed as an intersection of two matroids and the characterization of independent sets of switches is very similar to Hall's theorem [Hal35]. In this section we show that this similarity is no coincidence and that in fact, bipartite matching can be expressed in terms of global connectivity and that in this case the characterization of independent switch sets is exactly Hall's theorem.

Let $G = (A \cup B, E)$ be a bipartite graph with $|A| = |B|$. We construct a switch graph $G' = (V, S)$ as follows. Let V consist of B and a new vertex s. Now for each vertex $a \in A$ we create a switch $s_a = (s, N(a))$ where $N(a)$ denotes the neighbors of a in G. The graph G has a perfect matching if and only if G' has a connected configuration. This is the case if and only if all $|A|$ switches of G' are independent, that is, $|S'| < |V(S')|$ holds for every $S' \subseteq S$. By the construction of G' this is equivalent to the statement that every set $A' \subseteq A$ has at least $|A'|$ neighbors, which is Hall's theorem [Hal35].

SwitchConnect-T. We further briefly analyze a generalization of the global connectivity problem, where only a subset of *terminal* is required to be in the same connected component. The problem SwitchConnect-T is defined as follows. Given a switch graph $G = (V, S)$ and a set $T \subseteq V$ of terminals, does there exist a configuration c such that in G_c all vertices of T are in the same connected component?

Theorem 4.6. *SwitchConnect-T is NP-hard.*

Proof. We reduce from 3-Sat. Let φ be an instance of 3-Sat. We construct a switch graph G_φ as follows. For each clause C we create a vertex v_C and for each variable x we create two literal vertices v_x and $v_{\bar{x}}$. For each clause we create a switch $s_C = (v_C, T_C)$ where T_C is the set of vertices corresponding to literals that occur in C. Finally, we add one new node s and for each variable x a switch $s_x = (s, \{v_x, v_{\bar{x}}\})$. Let T be the set that contains all clause-vertices and the node s.

We claim that G_φ has a configuration that connects T if and only if φ has a satisfiable truth assignment. Given a truth assignment of φ we construct a configuration of G_φ as follows. For each variable x we set $c(s_x) = v_x$ if x has the value *true* and $c(s_x) = v_{\bar{x}}$ otherwise and for each clause C we set $c(s_C) = v$ where v is a vertex that corresponds to a satisfied literal of C. Conversely from a configuration c that connects T we can find a truth assignment by setting x to *true* if $c(s_x) = v_x$ and *false* otherwise.

The claim follows from the fact that a clause vertex v_C is connected to s if and only if the corresponding clause C is satisfied by the truth assignment. Note that we can reduce to binary switches by replacing each switch with three targets by two binary switches. □

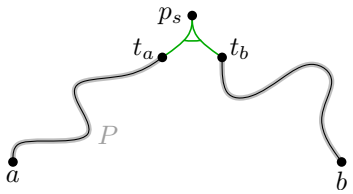
Although SwitchConnect-T is NP-hard in general it can be solved in polynomial time for some special cases. We have already shown that it can be solved efficiently in the case $T = V$. In the next section we will show that the problem can be solved in polynomial time if $|T| = 2$.

Moreover, from the polynomial-time algorithm for $T = V$ it follows that the problem is fixed-parameter tractable (FPT) with respect to the parameter $k' := n - |T|$ (that is whether it admits an algorithm with running time in $O(f(k')n^c)$, where c is a constant, and f is a function that depends only on k') since we can enumerate all possible subsets V' of $V \setminus T$ and solve the corresponding connectivity instance $G - V'$. It is an open question whether SwitchConnect-T is FPT with respect to $|T|$ or even if it can be solved in polynomial time if $|T| = 3$.

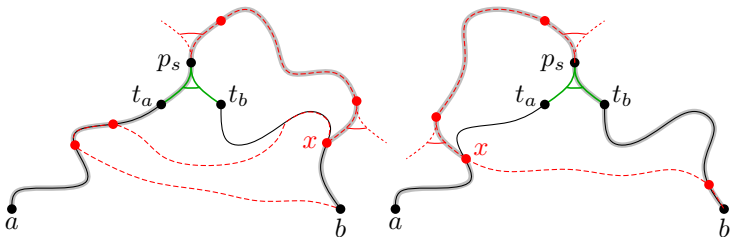
4.5. Local Connectivity

In this section, we investigate configurations that connect two given vertices a and b by a path. In the following, we call a sequence of switches a *forward path* if every switch's pivot is a target of its predecessor. A *contraction* of a switch s in a switch graph is defined as the switch graph identifying all vertices in $T_s \cup \{p_s\}$.

Lemma 4.2. *Let $G = (V, S)$ be a switch graph, a and b two vertices of G, and s be any switch such that in $(V, S \setminus \{s\})$, there is a (possibly trivial) forward path from p_s to b. Let G' be the result from contracting s. Then G can be switched to connect a and b if and only if this is possible for G'.*



(a) Non-trivial case: P (grey) connects both endpoints a and b with targets of s



(b) The forward path (dashed) from p_s to b hits a first vertex x on P, allowing to bypass either $p_s - t_b$ or $p_s - t_a$ without altering a switch on P.

Figure 4.6.: A path in $G'_{c'}$ witnesses a path in G_c for some configuration c.

Proof. First, by contracting a switch, it is not possible to lose connectivity. We will thus assume that it is possible to find a configuration c' that connects a and b in G' and show that this witnesses such a configuration c for G. We denote the path in $G'_{c'}$ as a sequence of switches P. If P forms a single path in $G_{c'}$, or if P connects either a or b to p_s, finding a connecting configuration is trivial. Otherwise, P forms two paths, connecting a to some $t_a \in T_s$ and b to some $t_b \in T_s$. In this situation, depicted in Figure 4.6, we can make use of the fact that in $S - s$, there is a forward path from p_s to b. Its first switch is not part of P, and we simply follow the forward path until we hit some vertex x on P. Now, switching the forward path from p_s to x gives us a bypass on either $p_s - t_a$ or $p_s - t_b$ and switching s accordingly connects a and b. □

This lemma provides a simple test for a-b-connectivity: If there is a configuration c that connects a and b in G_c, there either is a forward path from b to a or there is a contractable switch, since there must be a first switch that is used "forward" on the path from a to b in G_c. The proof of Lemma 4.2 is constructive, naively implemented, it yields an $O(n^2 + n\overline{m})$ time algorithm to test the existence of and compute a connecting configuration by storing a forward path for each contraction. It is not difficult to improve the running time for the problem of deciding whether an ab-path exists to almost linear time by using a Union-Find data structure. However, it is not as easy to also provide a corresponding path if it exists. Instead we show that the local connectivity problem is in fact equivalent to the problem of finding an augmenting path with respect to a matching. This yields a fast algorithm for solving the local connectivity problem.

Theorem 4.7. *Given a switch graph $G = (V, S)$ on n vertices whose underlying multigraph has $\overline{m}$ edges and two vertices $a, b \in V$, it can be determined in $O(\overline{m}+n)$ time whether a configuration that connects a and b exists.*

Proof. We construct a new graph $H = (V_H, E_H)$ and a matching of H as follows. For each vertex $v \in V$ we create two *node vertices* v_1 and v_2 in V_H and add to E_H the edge $e_v = v_1v_2$.

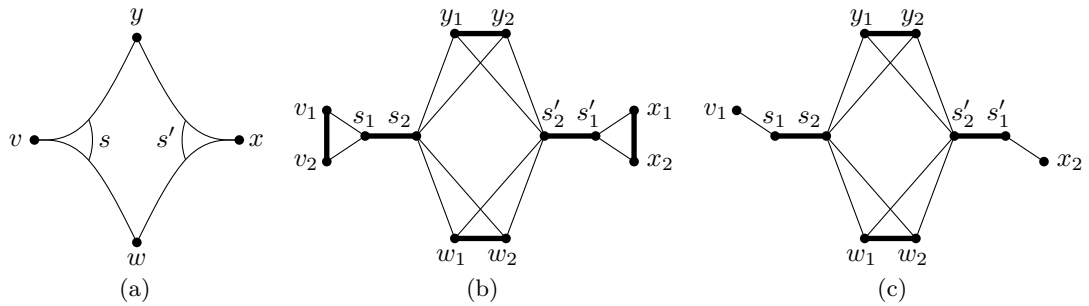


Figure 4.7.: Example for the reduction of local connectivity to finding an augmenting path. A switch graph (a) and corresponding graph H with matching M drawn as bold edges (b). The augmenting path $v_1 s_1 s_2 w_1 w_2 s_2' s_1' x_2$ in $H - \{v_2, x_1\}$ corresponds to the path vwx in G (c). The fact that $H - \{y_2, w_1\}$ does not admit a perfect matching shows that there is no path from y to w in G.

We call these edges *node edges*. For each switch $s = (p, T) \in S$ we create two *switch vertices* s_1 and s_2 in V_H and we add the edges $s_1 p_1$ and $s_1 p_2$, which we call *pivot connector edges*, the *switch edge* $s_1 s_2$, and for each of its targets $t \in T$ the edges $s_2 t_1$ and $s_2 t_2$, called *target connector edges*. We choose the matching $M = \{v_1 v_2 \mid v \in V\} \cup \{s_1 s_2 \mid s \in S\}$. We now claim that there is a configuration that connects a and b in G if and only if there exists an augmenting path from a_1 to b_2 in $H - \{a_2, b_1\}$ with respect to $M' = M \setminus \{a_1 a_2, b_1 b_2\}$. See Figure 4.7 for an example.

Let c be a configuration such that a and b are connected. Then a simple path between a and b in G_c can be described by an alternating sequence of vertices and switches $v^1 s^1 v^2 s^2 \dots v^{k-1} s^{k-1} v^k$ with $a = v^1$, $b = v^k$ and such that all switches and all vertices are distinct and for $i = 1, \dots, k-1$ it holds that $e_c(s_i) = v_i v_{i+1}$. To compute an alternating path between a_1 and b_2 in $H - \{a_2, b_1\}$ we drop all vertices of this sequence and replace each switch s^i for $i = 1, \dots, k-1$ as follows. If v^i is the pivot of s^i we replace s^i by $v_1^i s_1^i s_2^i v_2^{i+1}$, otherwise we replace it by $v_1^i s_2^i s_1^i v_2^{i+1}$. This substitution results in an alternating $a_1 b_2$-path in H with respect to M'.

Conversely assume that we have an alternating $a_1 b_2$-path in $H - \{a_2, b_1\}$ with respect to M'. A first observation is that any alternating path in H that contains a node vertex v_1 or v_2 stemming from a vertex $v \in V \setminus \{a, b\}$ must also contain the corresponding node edge $v_1 v_2$ and thus both node vertices since $v_1 v_2$ is in M'. The same holds for switch vertices s_1 and s_2 for all switches $s \in S$. Second, the matching edges that are contained in an alternating path must alternate between switch edges and node edges since by construction of H no two node vertices and no two switch vertices are connected by an edge that is not in M'. Further, for a node vertex and a switch vertex that are adjacent in H the corresponding switch is incident to the corresponding vertex in G. Hence, the alternating $a_1 b_2$-path in $H - \{a_2, b_1\}$ yields an ab-path in the underlying multigraph of G, that is, the graph that contains all edges that can possibly result from any configuration of G. As the path in H can contain at most one target connector edge of each switch this path contains at most one edge of each switch and hence can be realized by a configuration. This proves the claim.

It is not hard to see that the reduction can be performed in linear time and that also the resulting path can be translated back in linear time. The claim follows since the existence of an augmenting path can be checked in linear time [Tar83]. □

4.6. Even Degrees, Eulerian Graphs and Biconnectivity

In this section we study the problems of finding a Eulerian or a biconnected configuration and several related problems. A graph is Eulerian (that is it admits a cycle that uses each edge exactly once) if and only if it is connected and all vertices have even degrees. As we have seen in Section 4.4, a connected configuration of a switch graph can be found efficiently, if it exists. It turns out that a configuration for which all vertex degrees are even can be found efficiently, too.

Lemma 4.3. *For an undirected switch graph $G = (V, S)$, a configuration in which all vertex degrees are even (*SwitchEven*) can be computed in polynomial time.*

Proof. We use the results of Cornuéjols [Cor88] on the general factor problem: Let (W, E) be an undirected graph, and for every $v \in W$ let $D(v)$ be a subset of $\{1, \ldots, |W|\}$. Does there exist a subset $F \subseteq E$, such that in the graph (W, F) every vertex has its degree in $D(v)$? Cornuéjols [Cor88] shows that this problem can be decided in polynomial time, as long as the sets $D(v)$ do not contain any gap of length 2. (A set D of integers contains a gap of length 2, if it contains two elements d_1 and d_2, such that $d_2 \geq d_1 + 3$ and such that none of the numbers $d_1 + 1, \ldots, d_2 - 1$ is in D.)

For the proof of the lemma, construct a bipartite auxiliary graph between the set of switches and the set of vertices in the switch graph. Put an edge between any switch s and all targets in T_s. A vertex $v \in V$ is called odd (even), if it is the pivot of an odd (even) number of switches. For any switch $s \in S$ set $D(s) = \{1\}$. For any even vertex $v \in V$ set $D(v) = \{0, 2, 4, \ldots\}$, and for any odd vertex $v \in V$ set $D(v) = \{1, 3, 5, \ldots\}$. Note that none of these sets contains a gap of length 2. It can be seen that the auxiliary graph has a factor obeying the degree constraints if and only if the graph G has a configuration in which all vertex degrees are even. □

While this settles the membership of SwitchEven in $\mathcal{P}$, the algorithm is not very efficient. We therefore also provide a more elementary algorithm that is faster, namely needs time $O(n \cdot (\bar{m}+n))$. Let $G = (V, S)$ be a switch graph and let c be a configuration. We define a helper switch graph $H(c) = (V, S_c)$ that contains one switch for each switch of G in the following manner. For every $s \in S$ we define a corresponding switch $(c(s), T_s \setminus \{c(s)\})$ that has the current target of $c(s)$ as pivot and as targets the target set of s minus the current target. For ease of use we identify corresponding switches in the graphs G and $H(c)$.

Assume that two vertices a and b that are odd in G_c can be connected by a path in $H(c)$ with a configuration h. Let $s_1, \ldots, s_k$ be the set of switches in this path. We define a new configuration c' of G as follows: We set $c'(s) = h(s)$ if $s \in \{s_1, \ldots, s_k\}$ and $c'(s) = c(s)$ otherwise. Now the even vertices of $G_{c'}$ are exactly the even vertices of G_c plus a and b. By the definition of c' the degree of a vertex changes from G_c to $G_{c'}$ by 1 for each edge on the path between a and b in $H(c)_h$. Since all interior vertices of the path have an even number of incident edges only the parity of a and b changes. This suggests a very simple strategy for finding even configurations: Start with any configuration c of G and as long as there exists an odd vertex a find a path in $H(c)$ that connects a to an odd vertex b and change the configuration accordingly.

It remains to show that if the strategy does not succeed in finding an even configuration then there is none. We prove that if there exists an even configuration c^* of G then for any odd vertex a of G_c it is possible to switch a path in $H(c)$ that connects a to another odd vertex b. Consider the graph $H' = (V, E_{H'})$ with $E_{H'} = \{\{c(s), c^*(s)\} \mid s \in S \text{ with } c(s) \neq$

$c^*(s)\}$. Note that this graph can by definition be obtained as a subgraph of $H(c)$ with a suitable configuration h. Each edge in H' represents a change of the degree of its incident vertices by 1 when changing c to c^*. Therefore, even (odd) vertices of G_c have even (odd) degree in H'. Hence a must have odd degree in H'. Since the connected component of H' that contains a must have an even number of odd vertices, there is an odd vertex b in this component and hence we can switch a path between a and b in $H(c)$ as claimed. We have proved the following theorem.

Theorem 4.8. *For an undirected switch graph $G = (V, S)$,* SwitchEven *can be decided in $O(n \cdot (\bar{m} + n))$ time.*

As we have seen, we can efficiently check whether a given switch graph admits a connected configuration and whether it admits an even configuration. A Eulerian configuration is one that satisfies both properties simultaneously. Interestingly, this combined problem is much more difficult than the two individual problems, and in fact NP-hard as we show in the next theorem. Moreover, achieving higher degrees of connectivity, such as finding a biconnected configuration or a strongly connected configuration in the case of directed graphs is NP-hard as well.

Theorem 4.9. *For binary undirected switch graphs it is NP-hard to decide whether there is a Eulerian or a biconnected configuration. For forward directed switch graphs it is NP-hard to decide whether there is a Eulerian or a strongly connected configuration.*

Proof. We reduce from DirectedHamiltonianCycle, which is known to be NP-hard for directed graphs with out-degree bounded by two [Ple79].

Let $G = (V, E)$ be a directed graph with out-degrees 1 and 2. We define a switch graph $H = (V, S)$ as follows. For each vertex $v \in V$ we add a switch $s_v = (v, N(v))$ where $N(v) = \{u \in V \mid (v, u) \in E\}$. Now, since for every configuration c the graph H_c has n vertices and n edges, the following properties are equivalent:

(i) G has a directed Hamiltonian cycle.

(ii) H has a directed Eulerian configuration as a directed switch graph.

(iii) H has a strongly connected configuration as a directed switch graph.

(iv) H has a biconnected configuration as an undirected switch graph.

(v) H has a Eulerian cycle as an undirected switch graph.

The claim follows since the reduction can be performed in linear time. □

4.7. Acyclic and Almost Acyclic Graphs

This section mainly deals with forward directed switch graphs (as defined in Section 4.2): We check in polynomial time whether such a graph has a DAG configuration, and we show that finding a configuration with the minimum number of directed cycles is NP-hard.

Hence, let $G = (V, S)$ be a forward directed switch graph, and observe the following. First: The out-degree of every vertex in G_c is independent of the chosen configuration. Second: If all vertices in a digraph have out-degree at least 1, then the graph contains a

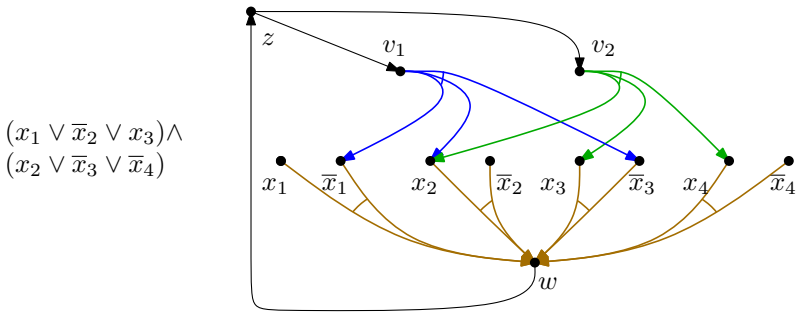


Figure 4.8.: Reduction of 3-Sat to SwitchDirectedAcyclic with reverse switches.

directed cycle. Third: If G contains a sink v (that is, a vertex v with out-degree 0), then it is safe to configure all switches s with $v \in T_s$ towards this sink. These three observations suggest a simple procedure: As long as the graph contains a sink v, we first set $c(s) := v$ for all switches s with $v \in T_s$, and then remove v together with all these switches. The procedure either stops with an empty graph (and an acyclic configuration), or with a non-empty subgraph of G in which all vertices have out-degree at least 1 (in which case there is no acyclic configuration). The algorithm can easily be implemented to run in linear time. In contrast, finding an acyclic configuration in general directed switch graphs and minimizing the number of cycles in forward directed switch graphs is hard.

Theorem 4.10. *For a forward directed switch graph, it can be decided in $O(n + \bar{m})$ time if it has an acyclic configuration. If an acyclic configuration exists, it can be found within the same time complexity.*

Theorem 4.11. *For a directed switch graph, it is NP-hard to decide if it has an acyclic configuration* (SwitchDirectedAcyclic).

Proof. The proof is by reduction from 3-Sat. Let φ be an instance of 3-Sat with variables $x_1, \dots, x_n$ and clauses $C_1, \dots, C_m$. We construct a switch graph G_φ as follows: We start with two vertices z and w and the arc (w, z). For each variable x_i we create two corresponding vertices $x_i, \overline{x}_i$ and a reverse switch $s_i = (\{x_i, \overline{x}_i\}, w)$. For each clause C_i we add a vertex v_i and the arc (z, v_i). Let x_u, x_v, x_w be the variables occurring in clause C_i. We set $\ell_u = x_u$ if x_u occurs negated in C_i and $\ell_u = \overline{x}_u$ otherwise. We define ℓ_v, ℓ_w analogously. We then add a *clause switch* $s_i^C = (v_i, \{\ell_u, \ell_v, \ell_w\})$. See Figure 4.8 for an example.

A satisfying truth assignment for φ yields an acyclic configuration c of G_φ: For each variable x_i we set $c(s_i) = x_i$ if x_i is assigned the value true and $c(s_i) = \overline{x_i}$ otherwise. Since each clause of φ is satisfied in this configuration at least one target of every clause switch has out-degree 0. Hence every clause switch can easily be configured to avoid all cycles.

Furthermore, an acyclic configuration c of G_φ yields a satisfying truth assignment for φ: We set variable x_i to true if $c(s_i) = x_i$ and to false otherwise. As the configuration is acyclic every clause switch must have a sink as target, and this sink represents a satisfied literal in the corresponding clause.

Note that although the clause switches have fan-out 3, the result also holds for binary switch graphs, as we can replace each switch with fan-out 3 by two binary switches without affecting the number of cycles with respect to any configuration. □

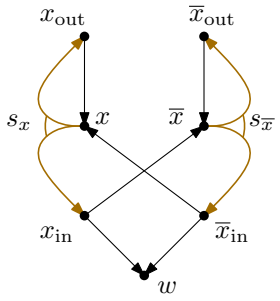


Figure 4.9.: Replacement of reverse switches for reduction of SwitchDirectedAcyclic to SwitchMinimumDirectedCycles

Theorem 4.12. *For a forward directed switch graph G and an integer $k > 0$, it is NP-hard to decide if there is a configuration with at most k cycles* (SwitchMinimumDirectedCycles).

Proof. We show how to simulate a binary reverse switches with usual binary forward switches at the cost of one additional cycle. Let G' be an instance of SwitchDirectedAcyclic with k binary reverse switches. We construct a directed switch graph G by replacing each reverse switch $s = (\{x, \overline{x}\}, w)$ by the following construction. We add four vertices $x_{\text{out}}, \overline{x}_{\text{out}}, x_{\text{in}}, \overline{x}_{\text{in}}$ along with the arcs $(x_{\text{in}}, \overline{x})$, $(\overline{x}_{\text{in}}, x)$, (x_{out}, x), $(\overline{x}_{\text{out}}, \overline{x})$ and the two forward switches $s_x = (x, \{x_{\text{in}}, x_{\text{out}}\})$, $s_{\overline{x}} = (\overline{x}, \{\overline{x}_{\text{out}}, \overline{x}_{\text{in}}\})$; see Figure 4.9.

As each replacement creates at least one cycle every configuration of G has at least k cycles. Each replacement has four distinct configurations. Two of them directly correspond to a configuration of the original reverse switch, namely the ones where one of the vertices $x, \overline{x}$ is connected to its in- and the other one to its out-vertex. We say that a configuration of G is *good* for the replacement in this case. There is a bijection between the acyclic configurations of G' and the configurations of G with k cycles that are good for each replacement.

Let c be a configuration of G with k cycles. We can modify c such that it is good for each replacement without increasing the number of cycles: The case $c(s_x) = x_{\text{out}}, c(s_{\overline{x}}) = \overline{x}_{\text{out}}$ can be excluded, as it would induce two cycles. In case $c(s_x) = x_{\text{in}}, c(s_{\overline{x}}) = \overline{x}_{\text{in}}$ we can change $c(s_x) := x_{\text{out}}$ without increasing the number of cycles thus making c good for the replacement. This operation does not increase the number of cycles, as we introduce at most one new cycle, namely x_{out} but at the same time remove at least the cycle $xx_{\text{in}}\overline{x}\overline{x}_{\text{in}}$.

Hence G admits a configuration with at most k cycles if and only if G' has an acyclic configuration. Since G can be constructed in linear time from G' the problem to decide whether such a configuration exists is NP-hard. □

4.8. Concluding Remarks

In this chapter we have studied the complexity of fundamental problems on switch graphs. While finding a configuration of a switch graph such that the resulting graph satisfies a certain property $\mathcal{P}$ is NP-hard for many properties, we gave efficient algorithms for certain connectivity problems. In particular, we have shown how to test efficiently, whether all vertices of a switch graph can be connected and how to check whether two given vertices can be connected.

One approach to solve the global connectivity problem was to consider two matroids on the edges, the subsets of edges forming an acyclic subgraph and the subsets of edges that can result from a configuration. The intersection of these two matroids contains exactly those acyclic edge sets that can be produced by configurations and hence its elements of size $n-1$ correspond to spanning trees that can be produced by configurations. This is a very general approach and shows that in fact for any graph property that induces a matroid on the edges (for example, rigidity [Lam70]), the corresponding switch graph problem can be solved efficiently.

Finding a planar configuration can be formalized similarly, one set contains again the edge sets that can result from configurations, while the other one contains the edge sets that result in a planar graph. Here the sets of size m in the intersection correspond to the planar configurations. However, the problem of determining whether a planar configuration exists turned out to be NP-hard. The crucial difference is that the planar subgraphs of a graph do not form a matroid; removing a few edges from a maximal planar edge set $E' \subseteq E$ may allow for inclusion of other edges of E, possibly resulting in an edge set E'' with $|E''| > |E'|$ that is still planar.

Open Problems. We leave open the question whether it is possible to check in polynomial time if three given vertices can be connected simultaneously and, more generally, whether the problem SwitchConnect-T is fixed-parameter tractable with respect to $|T|$.

Chapter 5

Matchings in Planar Graphs with Fixed Minimum Degree

In this chapter we present algorithms that compute large matchings in planar graphs with fixed minimum degree. The algorithms give a guarantee on the size of the computed matching and run in linear time. Thus they are faster than the best known algorithm for computing maximum matchings in general graphs and in planar graphs, which run in $O(\sqrt{n}m)$ and $O(n^{1.188})$ time, respectively. For the class of planar graphs with minimum degree 3 the bound we achieve is known to be best possible. Further, we discuss how minimum degree 5 can be used to obtain stronger bounds on the matching size.

The chapter is based on joint work with Robert Franke and Dorothea Wagner [FRW10].

5.1. Introduction

A *matching* is a set of independent (that is, pairwise non-adjacent) edges in a graph. A *maximum* matching is a matching of maximum cardinality, and a *maximal* matching cannot be enlarged by adding edges.

The problem of finding maximum matchings in graphs has a long history, dating back to Petersen's theorem [Pet91], which states that every biconnected 3-regular graph has a *perfect* matching, that is a matching that matches every vertex.

Finding maximum matchings, or large matchings in general, has many applications; see for example the book on matching theory of Lovász and Plummer [LP86]. To date the asymptotically fastest (but rather complicated) algorithm for finding maximum matchings in general graphs runs in $O(\sqrt{n}m)$ time [MV80], where n and m are the numbers of vertices and edges of the given graph, respectively. Only recently have faster algorithms for dense graphs, for planar graphs, for graphs of bounded genus, and for general H-minor free graphs been suggested. They are all based on fast matrix multiplication (which, as a tool, is not very practical) and run in $O(n^\omega)$ time for dense graphs [MS04], $O(n^{\omega/2})$ time for planar graphs [MS06] and for graphs of bounded genus [YZ07], and in $O(n^{3\omega/(\omega+3)}) \subset O(n^{1.326})$ time for H-minor free graphs [YZ07], where $\omega \leq 2.376$ is the exponent in the running time of the best known matrix-multiplication algorithm [CW87]. However, for practical purposes, often slower, but less complicated algorithms are used: both LEDA [Alg07] and the Boost Graph Library [SLL07] provide maximum-matching algorithms that are based on repeatedly finding augmenting paths and have a running time of $O(nm\,\alpha(n,m))$[Tar83].

There has been a sequence of increasingly general characterizations of graphs with

perfect matchings [Pet91, Hal35, Tut47]. This has also led to algorithms that test the existence of or compute perfect matchings in $o(\sqrt{n}m)$ time in, for example, bipartite k-regular graphs [Sch99, COS01], 3-regular biconnected graphs [BBDL01], and subgraphs of regular grids [Thu90, HZ93, KR96]. The last four algorithms all work in linear time for the corresponding subclasses of planar graphs. Moreover, for planar bipartite graphs a perfect matching can be computed in $O(n \log^3 n)$ time if it exists [MN95, FR06]. There is also a fast algorithm for finding unique maximum matchings [GKT01]. It takes $O(m \log^4 n)$ time in general and $O(n \log n)$ time in planar graphs.

There are combinatorial results that prove lower bounds on the size of maximum matchings in certain graph classes. Nishizeki and Baybars [NB79] show that planar graphs with minimum degrees 3, 4 and 5 have matchings of size at least $(n+2)/3$, $(2n+3)/5$ and $(5n+6)/11$, respectively. Biedl et al. [BDD+04] show that maxdeg-3 graphs have a matching of size $(n-1)/3$, 3-regular graphs have a matching of size $(4n-1)/9$ and 3-connected planar graphs have a matching of size $(n+4)/3$. However, these proofs are not constructive; in particular, they do not indicate a way to find such a matching faster than by computation of a maximum matching. The only simple way to exploit these bounds algorithmically is to use the fact that a maximal matching (which can be computed quickly) has at least half the size of a maximum matching. The bounds obtained in this way are, however, rather weak, for example, $(n+2)/6$ for planar graphs with minimum degree 3 compared to the tight $(n+2)/3$.

Recently, Rutter and Wolff [RW10] (a preliminary version appeared as [RW08c]) gave fast algorithms that achieve the tight bounds of Biedl et al. Their algorithms compute matchings of size $(n-1)/3$ in maxdeg-3 graphs in linear time, of size $(4n-1)/9$ in 3-regular graphs in $O(n \log^4 n)$ time and of size $(n+4)/3$ in 3-connected planar graphs in linear time. For graphs with bounded maximum degree k lower bounds for the size of maximal matchings have been considered [Han08].

Contribution. Unfortunately, none of the above results can be used to obtain matchings of guaranteed size in planar graphs with fixed minimum degree. In fact the question how fixed minimum degrees can be exploited algorithmically was posed as an open question in [RW10]. We answer this question and show that the tight bounds of Nishizeki and Baybars [NB79] for minimum degree 3 can be reached in linear time. We further analyze our algorithm in the context of minimum degree 5 and show that with some small modification it yields a matching of size $(2n+1)/5$ in this case.

The bound of Nishizeki and Baybars [NB79] for connected planar graphs with minimum degree 3 was also obtained by Papadimitriou and Yannakakis [PY81]. They analyze the structure of maximum matchings for these graphs and show that the structure is such that the free vertices can be balanced against the matching edges. We show that if we construct the matching accordingly, this balancing can be done locally: there is a pairing of free vertices with matching edges such that each free vertex is "adjacent" to its partner, in the sense that the free vertex is adjacent to at least one endpoint of the matching edge it is paired with.

Outline. The chapter is structured as follows. In Section 5.2 we propose a simple algorithm that already gives a non-trivial guarantee on the matching size, yet fails to reach the tight bound of $(n+2)/3$ for planar minimum degree 3 graphs. We then analyze the algorithm and come up with additional structural conditions for the matching that allow us to improve its size. Section 5.3 then shows how these structural constraints can be employed to obtain a linear-time algorithm that finds matchings of size $(n+2)/3$ in planar

graphs with minimum degree 3. We discuss how our approach can be generalized to obtain better bounds for planar graphs with minimum degree 5 in Section 5.4. We conclude and pose some open questions in Section 5.5.

5.2. Exploiting Minimum Degrees

In this section we describe a simple linear-time matching algorithm that already gives a non-trivial guarantee for planar mindeg-3 graphs. Our tight analysis then shows which aspects of the algorithm need to be improved in order to achieve the tight bounds of Nishizeki and Baybars [NB79].

We then show that certain additional structural requirements on the matching ensure that for minimum degree $\delta = 3$ we obtain the tight bound of Nishizeki and Baybars [NB79]. This analysis forms the basis of the algorithm presented in Section 5.3 where we show that a corresponding matching can be found quickly.

In order to present the algorithms we need some standard notation for graphs and matchings. Let $G = (V, E)$ be a graph and let M be a matching of G. A vertex in V is *free* (with respect to M) if it is not incident to an edge of M. An *augmenting path* P (with respect to M) is a path that alternates between edges in M and edges in $E \setminus M$ and starts and ends at different free vertices. In this case the symmetric difference of P and M is a matching of size $|M| + 1$. A matching is *k-free* if it does not admit an augmenting path of length up to k.

5.2.1. Algorithm Based on Short Augmenting Paths

We propose the following two-step algorithm Match3Aug:

(1) Compute a maximal matching.

(2) Iteratively find augmenting paths of length 3.

Lemma 5.1. *Let $G = (V, E)$ be a connected graph with m edges.* Match3Aug *computes a 3-free matching in $O(m)$ time.*

Proof. Step 1 is performed by choosing edges greedily. For Step 2 it is sufficient to consider the matching edges one by one and to check whether they are contained in an augmenting path of length 3. For an edge uv this can be done in $O(d(u) + d(v))$ time. The overall linearity follows from the fact that edges that are added to the matching during Step 2 need not be checked. Let xy be an edge that is added in Step 2 such that x was free after Step 1. If xy was contained in an augmenting path of length 3, then x would have a free neighbor x' contradicting the maximality of the matching after Step 1. □

In the following, we analyze the size of 3-free matchings in planar graphs with minimum degree δ. To this end, we divide the free vertices into two disjoint sets that we bound independently.

Let $G = (V, E)$ be a planar graph with minimum degree δ and let M be a 3-free matching. Let $e \in M$ be an edge such that there is a free vertex $v \in V$ that is adjacent to both endpoints of e. We say that v *covers* e and that e is *covered*. An edge of the matching that is not covered by a vertex is *open*. Let M_C and M_O denote the set of covered and open edges of M, respectively. Moreover, let F_C denote the set of vertices that cover an edge

and let F_O be the set of free vertices that do not cover any edge. Note that, by definition, M_C and M_O form a partition of M, and F_C and F_O form a partition of the free vertices of V. Hence we have that $|M| = |M_C| + |M_O|$ and $n = 2 \cdot |M| + |F_C| + |F_O|$. We now bound the number of free vertices by independently bounding $|F_C|$ and $|F_O|$.

Lemma 5.2. *Let $G = (V, E)$ be a planar graph with minimum degree δ, let M be a 3-free matching and let M_C,M_O, F_C and F_O be defined as above. Then,*

$$|F_C| \leq |M_C| \tag{5.1}$$

$$|F_O| \leq 2 \cdot \frac{|M_O| - 2}{\delta - 2}. \tag{5.2}$$

Proof. First note that Equation (5.1) holds since the vertex covering an edge is unique as there would be an augmenting path of length 3 otherwise.

For the proof of Equation (5.2) consider the bipartite auxiliary graph $G' = (V', E')$ whose vertices are the vertices in F_O and the open edges of M. We connect a vertex $v \in F_O$ with an edge $m \in M_O$ if v is adjacent to an endpoint of m in G. The graph G' is planar as it can be obtained as a minor of G by removing all vertices that are either incident to an edge in M_C or cover an edge, contracting the remaining matching edges and removing edges that are not incident to a free vertex. Since no vertex of F_O is adjacent to an endpoint of a covered edge (there would be an augmenting path of length 3 otherwise), each vertex in F_O has degree at least δ in G'. Equation (5.2) now follows from $|E'| \leq 2 \cdot |V'| - 4 = 2 \cdot (|F_O| + |M_O|) - 4$ (bipartite, planar) and $|E'| \geq \delta |F_O|$ (minimum degree). □

Theorem 5.1. *Let $G = (V, E)$ be a planar graph with n vertices, minimum degree $\delta \in \{3, 4\}$ and let M be a 3-free matching. Then the following holds:*

$$|M| \geq \frac{(\delta - 2) \cdot n + 4}{2 \cdot (\delta - 1)}. \tag{5.3}$$

Proof. This follows from Lemma 5.2 and $n = 2 \cdot |M| + |F_O| + |F_C|$. □

Equation (5.3) does not hold for $\delta = 5$ as in this case the bound on $|F_C|$, which is independent of δ, is too weak. By Theorem 5.1 Match3Aug computes in linear time matchings of size at least $(n+4)/4$ in planar graphs with minimum degree 3 and matchings of size $(n + 2)/3$ in planar graphs with minimum degree 4.

In order to obtain the bound $(n + 2)/3$ for $\delta = 3$ we would like to improve the bound of $|F_O|$ from Equation (5.2). However, the example in Figure 5.1 shows that our analysis is tight. Roughly speaking the problem is that the graph induced by the matching in this example is not connected. In the next section we give a precise definition of the desired property of the matching in order to improve the bound on the size.

5.2.2. More Structure via Pure Tree-Like Matchings

Let $G = (V, E)$ be a planar graph with a fixed planar embedding, that is, for every vertex v we have a cyclic ordering $\sigma(v)$ of its incident edges, and let M be a matching of G. Let G_M be the graph that is induced by the matched vertices of M. A matched vertex v is *cyclically pure* if its incident edges in G_M form an interval in $\sigma(v)$; further, M is called *pure* if all matched vertices are cyclically pure. The matching M is called *tree-like* if G_M is a tree.

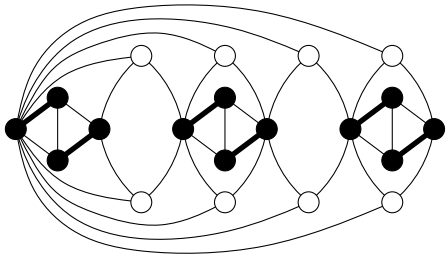
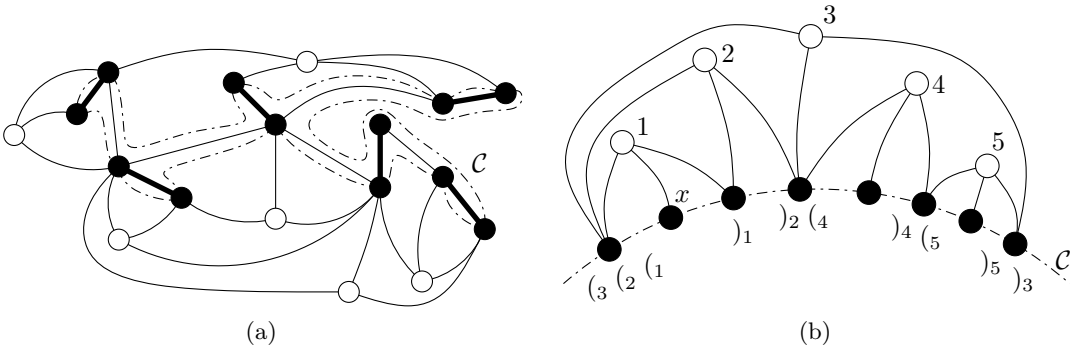


Figure 5.1.: Planar graph with n vertices, mindeg 3 and a 3-free matching with only $(n+4)/4$ edges.



(a) (b)

Figure 5.2.: (a) Construction of curve $\mathcal{C}$ that separates matched and free vertices of a pure tree-like matching. (b) Parenthetical structure of vertices in F_M along $\mathcal{C}$. The inner neighbors of the most interior vertices of F_M have a unique neighbor in F_M, for example, vertex 1 is the unique neighbor of x.

Lemma 5.3. *Let $G = (V, E)$ be a planar embedded graph and let M be a pure tree-like matching in G such that all free vertices have degree at least δ. Let F_M be the set of free vertices that have only matched neighbors. If F_M is not empty then there is a vertex $v \in F_M$ that has matched neighbors $x_1, \ldots, x_{\delta-2}$ such that each x_i has no other neighbor in F_M.*

Proof. In this proof we distinguish between *outer* vertices, that is, matched vertices with free neighbors and *inner* vertices, that is, matched vertices that are not outer. To prove the lemma we consider the subgraph G' of G that is induced by the edges that have one endpoint in F_M. We show that all outer vertices share a common face in the embedding inherited from G. Hence, by planarity, the vertices in F_M must have a parenthetical structure, where the most interior ones have the desired property.

Assume we have a planar drawing of G that realizes its given embedding. We construct a simple closed curve $\mathcal{C}$ that contains all outer vertices, encloses all inner ones and separates the matched vertices from the free vertices.

To construct $\mathcal{C}$, we pick an arbitrary outer vertex v as starting point, traverse the Euler tour of the tree G_M induced by the embedding and draw the curve along the edges of G_M. Beginning from v, we traverse G_M, starting along one of the edges bounding the interval of edges that connect v to a vertex in F_M. We position an imaginary pencil on the point where v lies in the plane and draw along the edges of G_M. Whenever we reach an outer vertex v' and would have to cross an edge that is incident to a free vertex to pass it, we instead draw right through v' and thus separate the free neighbors of v' from the matched

ones. Since all vertices are cyclically pure the line visits every vertex at most once. Hence $\mathcal{C}$ is a simple cycle that contains every outer vertex. By construction $\mathcal{C}$ does not cross any edge and encloses all inner vertices but none of the free vertices; see Figure 5.2a. Removing the interior of $\mathcal{C}$ (including edges) and free vertices that are not in F_M yields G' with all outer vertices sharing a common face.

There is at least one outer vertex that is incident with the outer face of G'. Since all vertices in F_M are outside $\mathcal{C}$ there is at least one vertex x in F_M that is incident to the outer face of G'. Let e be an edge that is incident to x and bounds the outer face. The other endpoint of e is the desired vertex as it is matched and hence belongs to $\mathcal{C}$. Let $b_1, \dots, b_k$ be the outer vertices as they occur along $\mathcal{C}$ in clockwise order, where b_1 is incident with the outer face of G_M.

Now consider the vertices of F_M (note that all their neighbors belong to $\mathcal{C}$). For each vertex v in F_M we set $b_\ell(v)$ and $b_r(v)$ to be the vertex b_i with the smallest, respectively largest, index i such that b_i is incident to v. We attach an opening parenthesis with label v to the edge $\{b_\ell(v), v\}$ for each v in F_M. Analogously we attach a closing parenthesis with label v to the edge $\{b_r(v), v\}$. We then traverse $b_1, \dots, b_k$ and collect at each vertex all parentheses in clockwise order; see Figure 5.2b. This yields a sequence of opening and closing parentheses where matching opening and closing parentheses have the same label since a structure like $({}_a\,({}_b\,)_a\,)_b$ would contradict planarity.

Now pick a pair of parentheses that does not enclose any other parentheses. Let v be the vertex that induces this pair. In addition to the two matched neighbors at which the parentheses were placed there have to be at least $\delta - 2$ further matched neighbors $x_1, \dots, x_{\delta-2}$ since the degree of v is at least δ. Each of these vertices has only one free neighbor in F_M, namely v. □

This result on pure tree-like matchings can be used to improve the bound on $|F_O|$ and hence the bound on 3-free matchings.

Lemma 5.4. *Let $G = (V, E)$ be a planar graph with minimum degree δ, let M be a pure tree-like 3-free matching and let M_O and F_O be defined as above. Then,*

$$|F_O| \leq \frac{|M_O| - 2}{\delta - 2}. \tag{5.4}$$

Proof. Let $\mathcal{C}$ be the curve from the proof of Lemma 5.3. Similar to the proof of Lemma 5.2 let G' be the graph obtained from G by contracting the edges in M_O, removing the vertices in F_C, the endpoints of edges in M_C and all edges in the interior of $\mathcal{C}$. This again yields a bipartite planar graph where one of the vertex sets corresponds to the open matching edges. Note that $\mathcal{C}$ is still a simple cycle since $\mathcal{C}$ contains at most one endpoint of each edge in M_O. However, since the interior of $\mathcal{C}$ is empty, the graph obtained from duplicating the vertices in F_O (together with their incident edges) is still planar and bipartite. Thus the bound is improved by a factor of 2, yielding the claim. □

With the stronger bound of Equation 5.4 it follows that a pure tree-like 3-free matching in a planar graph with n vertices and minimum degree 3 has size at least $n/3$. For minimum degrees 4 and 5 the bound on $|F_C|$ is now weaker than the bound on $|F_O|$. Hence to obtain even stronger bounds we would need to improve the bound on the size of F_C.

Unfortunately, it is not easily possible to find a maximal matching that is both pure and tree-like in a given graph. Instead we show that we can construct such a matching in a subgraph of the input graph, by carefully removing free vertices when we cannot continue

with enlarging the matching. The main part is to show that the number of removed vertices is bounded by the number of matching edges.

5.3. Algorithm

In this section we describe an algorithm that computes in linear time a matching of size at least $(n+2)/3$ in planar graphs with minimum degree 3. To show that our algorithm actually finds a matching of this size we use the following argument. In the course of the algorithm we perform a series of steps, each of which either increases the size of the matching by 1 or deletes a free vertex. However, whenever a vertex is deleted, we make sure that there is an edge in the matching that "remembers" it in such a way that each matching edge "remembers" at most one vertex and no vertex is ever "forgotten". The algorithm finishes when there are no free vertices left. The bound then follows from the observation that there can be at most as many free vertices as matching edges.

The algorithm works as follows. We start by adding an arbitrary edge to the matching, which clearly is both pure and tree-like. We then enlarge the matching and make sure it remains pure and tree-like. To find an adequate spot to try to enlarge the matching we use Lemma 5.3. If there are only free vertices that also have free neighbors (that is, $F_M = \emptyset$), we can easily find an edge that can be used to enlarge the matching; see Section 5.3.1. If F_M is not empty (that is, there are free vertices that have only matched neighbors) the lemma yields a free vertex v and a matched vertex x such that v is the only neighbor of x in F_M. In this case we try to enlarge the matching by two different strategies.

(a) If x has free neighbors that have further free neighbors, we will use one of these and add an edge between two free vertices to the matching (Section 5.3.1).

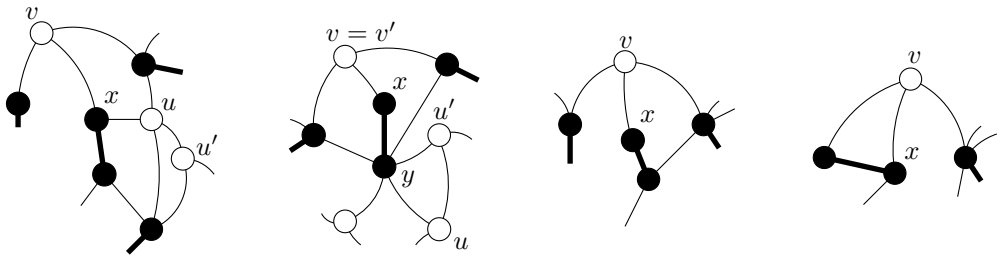
(b) If there is an augmenting path $vxyu$ of length 3, we will use this fact to swap xy for two new matching edges (Section 5.3.2).

When neither of these strategies can be applied we remove v and show that there is a suitable matching edge that can remember it. The algorithm stops when no free vertices are left. In the following sections we describe these steps in detail and prove that they preserve a pure tree-like matching.

5.3.1. Enlargement by Adding a Suitable Edge

In this section we discuss how to enlarge a pure tree-like matching M by adding a suitable edge such that the outcome is still pure and tree-like. Consider a matched vertex x that has free neighbors and some of these have further free neighbors. Since the edges that connect x to free vertices form an interval in $\sigma(x)$, there exist a leftmost and a rightmost free neighbor of x (they coincide if x has only one free neighbor). To preserve cyclic purity we need that the leftmost or rightmost free neighbor u of x has a free neighbor u'. This situation occurs if x has at most one free neighbor that belongs to F_M and x is adjacent to a free vertex that is not in F_M; see Figure 5.3a. The exact procedure is shown in the proof of the following lemma.

Lemma 5.5. *Let $G = (V, E)$ be a planar graph and let M be a pure tree-like matching in G such that each free vertex has degree at least δ. Further let x be a matched vertex such that the leftmost or rightmost free neighbor of x is adjacent to a free vertex. Then*



(a) (b) (c) (d)

Figure 5.3.: Illustration of the different cases that can occur in the algorithm for the candidate vertex x and its unique neighbor v in F_M.

there is a graph $G' = (V, E')$ with $E' \subseteq E$ and a pure tree-like matching M' of G' such that $|M'| = |M| + 1$ and each free vertex has degree at least δ in G'.

Proof. Without loss of generality, we can assume that the leftmost free neighbor u of x has a free neighbor. We now scan $\sigma(u)$ beginning with x until we find the first free neighbor u'. Let M' be $M \cup \{uu'\}$ and let G' be the graph that we obtain from G by removing all edges between u or u' and another matched vertex except for xu and uu'. We show that G' and M' satisfy the claim.

First, it is obvious that $|M'| = |M|+1$ holds and each free vertex has the same degree as before since we only deleted edges that have both endpoints matched. It remains to show that M' is pure and tree-like. The vertex x is cyclically pure since u was the leftmost free neighbor of x and a possible edge xu' has been removed. Vertex u is cyclically pure since it has just two matched neighbors x and u' and the edges ux and uu' are adjacent in $\sigma(u)$. Vertex u' is cyclically pure, because u is its only matched neighbor. The other matched vertices remain also cyclically pure as removing edges never violates cyclic purity. Thus M' is pure. Moreover, M' is tree-like since $G'_{M'}$ can be obtained by adding the branch xuu' to G_M (u and u' were free and thus not in G_M). □

5.3.2. Exploiting Existence of an Augmenting Path of Length 3

In this section we describe how to make use of an augmenting path of length at most 3 in our context. Let $G = (V, E)$ be a planar embedded graph, let M be a pure tree-like matching in G such that all free vertices have degree at least 3 and let $vxyu$ be an augmenting path of length 3. We show that we can modify G and M such that M is enlarged by 1 and remains pure and tree-like. The problem is that just using the augmenting path to enlarge the matching may violate the cyclic purity of the vertices u, v, x and y; an example of such a situation is shown in Figure 5.3b. Instead we show that there exists a suitable augmenting path of length 3 which leads (after removing some edges whose endpoints are both matched) to an enlarged pure tree-like matching.

Lemma 5.6. *Let $G = (V, E)$ be a planar graph and let M be a pure tree-like matching in G such that each free vertex has degree at least δ. Let $vxyu$ be an augmenting path of length 3. Then there is a graph $G' = (V, E')$ with $E' \subseteq E$ and a pure tree-like matching M' such that $|M'| = |M| + 1$ and each free vertex has degree at least δ in G'.*

Proof. Let x_ℓ and x_r be the leftmost and rightmost free neighbor of x, respectively, and define y_r,y_ℓ analogously. Choose $v' \in \{x_\ell, x_r\}$, $u' \in \{y_\ell, y_r\}$ such that v' and u' are distinct. This is always possible, otherwise $x_\ell = x_r = y_\ell = y_r$ and x and y both have only one free neighbor, which is actually shared by x and y, contradicting $v \neq u$.

We set $M' := (M \setminus \{xy\}) \cup \{v'x, yu'\}$ and let G' be the graph obtained from G by removing all edges that connect v' or u' to a matched vertex other than their matching partner. Clearly $|M'| = |M| + 1$ holds. We claim that M' is a pure tree-like matching in G' and every free vertex of G' has degree at least δ.

First note that by the choice of v' and u' the edges $v'x$ and yu' do not violate the cyclic purity of x and y. The only edges that might violate the cyclic purity of x and y are the edges $v'y$ and xv'. They are however removed when going from G to G'. Hence x and y are cyclically pure. By construction of G', v' and u' each have only one matched neighbor hence they are cyclically pure as well. All vertices different from v', x, y, u' may only have lost edges to free neighbors in G. Hence they all remain cyclically pure and M' is pure.

The graph $G'_{M'}$ is a tree since it can be obtained from the tree G_M by inserting the edges $v'x$ and $u'y$, which add the new leaves v' and u'. Thus M' is tree-like. Since we only remove edges that are not incident to free vertices the degrees of all free vertices are preserved. □

5.3.3. Linear-Time Algorithm

Lemma 5.5 and Lemma 5.6 yield together with Lemma 5.3 the simple algorithm MatchMinDeg3, whose structure was outlined at the beginning of this section. A pseudo-code description of the algorithm is shown as Algorithm 5.1.

Theorem 5.2. *Let G be a planar embedded graph with n vertices and minimum degree 3. The algorithm* MatchMinDeg3 *computes a matching of size at least $(n+1)/3$ in $O(n)$ time.*

Proof. We begin this proof by stating and justifying some loop invariants for the while-loop in MatchMinDeg3.

(a) Each removed vertex is remembered by an adjacent matching edge

(b) Each matching edge remembers at most one vertex.

Algorithm 5.1: MatchMinDeg3

```
Select an arbitrary edge e and set M ← {e}
while there are still free vertices do
    if F_M ≠ ∅ then
        Select a matched vertex x and a free vertex v according to Lemma 5.3;
        if x has a free neighbor outside of F_M then
            The leftmost or rightmost free neighbor of x suits to apply Lemma 5.5
        else if there is an augmenting path vxyu then
            Enlarge the matching according to Lemma 5.6
        else
            Remove v (the matching edge that is incident to x remembers v)
    else
        Select a matched vertex that has free neighbors and apply Lemma 5.5
```

(c) The matching is pure and tree-like.

(d) Each free vertex has degree at least 3.

Invariants (a) and (b) are needed to prove the correctness of the algorithm while Invariants (c) and (d) ensure that the conditions of Lemmas 5.3, 5.5 and 5.6 are satisfied.

We now show that the algorithm preserves the invariants. When we delete a free vertex v, it is remembered by a matching edge xy (x and v are adjacent) that is not part of an augmenting path of length 3 and x has no other adjacent free vertices. Thus also y cannot be adjacent to a free vertex apart from v since there would be an augmenting path of length 3 otherwise. Hence xy will not have to remember another vertex and it is never removed from the matching; see Figures 5.3c and 5.3d. Thus Invariants (a) and (b) hold throughout the algorithm. Invariant (c) holds since we change the matching only by using Lemmas 5.5 and 5.6, which preserve the invariant. These lemmas together with the fact that we exclusively remove vertices that have only matched neighbors guarantee Invariant (d).

The size of the computed matching can now be seen as follows. Invariants (a), (b) and the observation that the last removed vertex has an additional remembering edge yield the bound $|F| \leq |M| - 1$ where F is the set of free vertices of G with respect to the output matching M. Using the equation $|F| = n - 2 \cdot |M|$ yields the bound $(n+1)/3 \leq |M|$.

Next, we discuss how to implement MatchMinDeg3 in linear running time. In each iteration the number of free vertices is decreased by at least 1. Thus the algorithm stops after at most n iterations. Next, we show that each iteration of the while-loop runs in amortized $O(1)$ time.

For each vertex we store whether it is matched and if it has free neighbors. When a vertex v becomes matched it requires $O(d(v))$ time to propagate this information to its neighbors such that they can update their number of free neighbors. The overall time spent in this step is linear since a matched vertex remains matched (although its matching partner may change).

For matched vertices with free neighbors, we additionally store the first and the last edge leading to a free vertex. Note that given a matching edge we can hence easily check whether it is part of an augmenting path of length 3 since this involves only a constant number of vertices, which can be found quickly using the first and last edge information. Note that the first and last edge can be updated in constant time when we remove an edge or match a free vertex. Moreover, for each matched vertex we store its F_M*-degree*, that is, its number of neighbors in F_M. When the last free neighbor of a free vertex v gets matched or v is deleted, v notifies its neighbors, which then update their F_M-degree. Both cases occur at most once for each free vertex and thus this notification work needs linear time in total. By keeping a list of vertices with F_M-degree 1 we can find a candidate vertex x as in Lemma 5.3 in constant time.

The check whether Lemma 5.5 or Lemma 5.6 can be used to enlarge the matching can be done in $O(1)$ time. We only need to check a constant number of vertices for membership in F_M, which can be done by storing whether a vertex has free neighbors or not. The vertices we need to check can easily be addressed via the leftmost and rightmost free neighbor pointers.

The total running time for all applications of the procedures provided by Lemma 5.6 and Lemma 5.5 is linear. This can be seen by considering occurrences of these cases. For applying Lemma 5.5 we first have to scan the neighborhood of a free vertex u for a free neighbor v, which requires $O(d(u))$ time. For the application of Lemma 5.6 the two vertices

u and v that are newly matched can be identified in $O(1)$ time. In both cases in order to ensure the cyclical purity for the two newly matched vertices u and v we scan $\sigma(u)$ and $\sigma(v)$, which requires $O(d(u) + d(v))$ time. Since u and v are matched afterwards, they will not be processed in the same way again. Finally, removing a free vertex v can also be done in $O(d(v))$ time. □

Note that we can miss the tight bound of $(n + 2)/3$ by 1. We can, however, enlarge the matching by 1 in $O(n)$ time by computing an augmenting path [Tar83].

5.4. A Better Bound for Minimum Degree 5

In this section we show how to improve the bound for $\delta = 5$ by ensuring that each removed vertex is remembered by more than one matching edge.

When F_M was not empty, we previously considered a matched vertex x that had only one neighbor v in F_M. This is the part where we make a small extension. Now, instead, we consider more such matched vertices with the same neighbor in F_M at once. Either one of them can be used to enlarge the matching via Lemmas 5.5 and 5.6 or none of them and their matching partners are adjacent to another free vertex. In this case all their matching edges can be used to remember v. For $\delta = 5$ Lemma 5.3 yields a free vertex v that has three such neighbors and hence at least two matching edges can remember v.

Theorem 5.3. *Let $G = (V, E)$ be a planar graph with n vertices and minimum degree 5. A matching of size at least $(2n + 1)/5$ can be computed in $O(n)$ time.*

Proof. Let M be the matching computed by the modified algorithm. As shown above each free vertex F is remembered by at least two matching edges. Together with the observation that the last free vertex is remembered by at least three edges we get $2 \cdot |F| + 1 \leq |M|$. The bound then follows from $n = 2 \cdot |M| + |F|$.

Next we show that the modified algorithm still runs in linear time. We need a way to maintain the set of candidate vertices with $\delta - 2$ matched neighbors according to Lemma 5.3. Instead of the F_M-degree of a matched vertex (that is, the number of neighbors in F_M) we store a list of its neighbors in F_M. Whenever the length of this list changes to 1 or from 1 to another number we notify the (previously) last neighbor. This notification enables vertices in F_M to keep track of their neighbors with F_M-degree 1 by maintaining a list of them. Hence we can maintain a list of vertices in F_M that have at least $\delta - 2$ neighbors with F_M-degree 1. Thus we can pick such a candidate in $O(1)$ time and try to enlarge the matching using $\delta - 2$ of its F_M-degree 1 neighbors. The procedures given by Lemmas 5.5 and 5.6 are called at most $\delta - 2$ times per iteration. □

5.5. Concluding Remarks

In this chapter, we have shown that it is possible to exploit minimum degrees in planar graphs algorithmically to obtain algorithms that compute matchings of guaranteed size quickly. Our algorithms run in linear time and yield matchings of size at least $(n + 2)/3$ and $(2n + 1)/5$ for planar graphs with minimum degrees 3 and 5, respectively. For planar mindeg-3 graphs this bound is best possible.

In this chapter we experienced a very different aspect of planarity than in the previous two chapters. In general, there exist graphs with minimum degree 3 that have no matching of size $(n+2)/3$, or even something close to this number. The graph $K_{3,n}$ for $n > 3$ has a maximum matching of size 3 and thus not even any constant fraction of the vertices can be matched in general mindeg-3 graphs. Thus planarity lies at the starting point of the problem we considered in this chapter. But this is not the only way in which we exploit planarity. Initially, we tried to come up with an algorithm that could in principle be run on any mindeg-3 graph and would yield the bound of Nishizeki et al. on planar mindeg-3 graphs. The algorithm Match3Aug from Section 5.2 is an example of such an algorithm, which does however provide a weaker bound on the matching size. We did not succeed in finding such an algorithm that achieves the tight bound. Our algorithm MatchMinDeg3 from Section 5.1 makes use of additional structural information provided by planarity. Namely, we use the inherent structural information that is encoded in a planar embedding of a planar graph to guide the algorithm in finding a matching that meets the desired bound. Since the algorithm requires that the edges around each vertex are ordered according to a planar embedding, the algorithm can only be run on planar graphs.

Open problems. While the bound $(n+2)/3$ for the matching size is tight for planar graphs with minimum degree 3, it is known that planar graphs with minimum degrees 4 and 5 admit matchings of size $(2n+3)/5$ and $(5n+6)/11$, respectively. Higher degrees of connectivity yield even better bounds. We leave open the question whether these tight bounds can be achieved in linear time.

It seems however, that this requires new insights and ideas that go beyond the concepts presented in this chapter. To achieve the tight bound of $(2n+3)/5$ for planar mindeg-4 graphs it is not sufficient to apply the algorithm for planar mindeg-3 graphs and to conduct a tighter analysis. There exist planar mindeg-4 graphs that have a pure, 3-free matching for which the graph induced by the matched vertices is connected but that do not reach the bound of $(2n+3)/5$. The construction of a family of such graphs G_i together with matchings M_i for $i \geq 0$ is illustrated in Figure 5.4. Figure 5.4a shows the graph G_0 on seven vertices together with a matching M_0 consisting of three edges. The graph G_i for $i > 0$ is constructed by joining three copies of G_{i-1} and adding an additional free vertex as shown in Figure 5.4b. Note that G_i has minimum degree 4 for $i > 0$. Further the matching M_i is pure and the graph G_M is connected (and hence can be made tree-like without reducing the degree of free vertices). We denote by n_i the number of vertices of G_i. By construction we have $n_0 = 7$ and $n_i = 3n_{i-1} - 3$ for $i > 0$. For the matching size we have $|M_0| = 3$ and $|M_i| = 3|M_{i-1}| - 2$. Solving these recurrences yields $n_i = (11 \cdot 3^i + 3)/2$ and $|M_i| = 2 \cdot 3^i + 1$. Hence we get that $|M_i|/n_i$ tends to $4/11 \approx 0.36$ for $i \to \infty$, thus missing the bound of $2/5 = 0.4$ proved by Nishizeki et al. [NB79].

Therefore we believe that additional algorithmic ingredients are required to achieve the tight bounds for graphs with minimum degrees 4 and 5. One such ingredient might be to try to do the balancing of free vertices against matching edges slightly less local by using, say, augmenting paths of length up to 5.

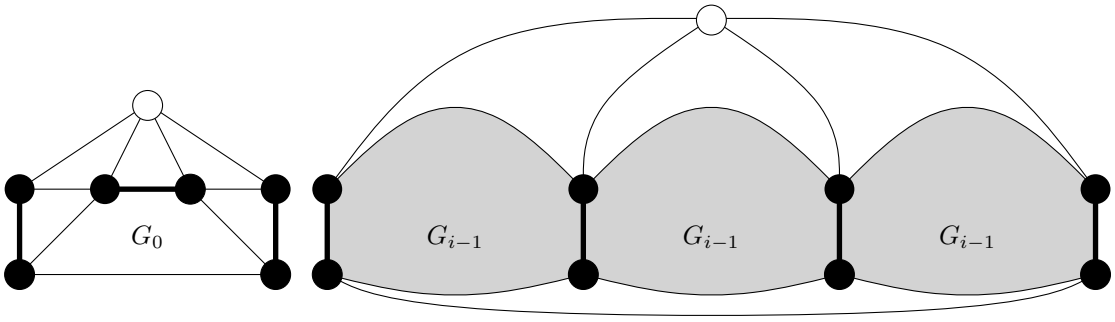


Figure 5.4.: Construction of a family of planar mindeg-4 graphs G_i together with pure, tree-like, 3-free matchings M_i of size less than $(2n+3)/5$. The graph G_0 (a) and the construction of G_i from three copies of G_{i-1} for $i > 0$ (b).

Part II.

Embeddings of Planar Graphs

Chapter 6

Testing Planarity of Partially Embedded Graphs

In this chapter we study the following problem: Given a planar graph G and a planar drawing (embedding) of a subgraph of a graph G, can such a drawing be extended to a planar drawing of the entire graph G? This problem fits the paradigm of extending a partial solution to a complete one, which has been studied before in many different settings. Unlike many cases, in which the presence of a partial solution in the input makes hard an otherwise easy problem, we show that the planarity question remains polynomial-time solvable. Our algorithm is based on several combinatorial lemmas, which show that the planarity of partially embedded graphs meets the "oncas" behaviour – obvious necessary conditions for planarity are also sufficient. These conditions are expressed in terms of the interplay between (a) rotation schemes and containment relationships between cycles and (b) the decomposition of a graph into its connected, biconnected, and triconnected components. This implies that no dynamic programming is needed for a decision algorithm and that the elements of the decomposition can be processed independently.

Further, by equipping the components of the decomposition with suitable data structures and by carefully splitting the problem into simpler subproblems, we make our algorithm run in linear time.

Finally, we consider several generalizations of the problem, for example, minimizing the number of edges of the partial embedding that need to be rerouted to extend it, and argue that they are NP-hard. Also, we show how our algorithm can be applied to solve related graph-drawing problems.

The chapter is based on joint work with Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, and Maurizio Patrignani [ADF+10a].

6.1. Introduction

In this chapter we pose and study the question of planarity testing in a constrained setting, namely when part of the input graph is already drawn and cannot be changed. A practical motivation for this question is, for example, the visualization of large networks in which certain patterns are required to be drawn in a standard way. The known planarity testing algorithms, even those that build a drawing incrementally, are of no help here since they are allowed to redraw at each step the part of the graph processed so far. For similar reasons, online planar embedding and planarity testing algorithms, such as [Wes92, Tam96, DT96a, Pou94], are not suitable to be used in this context.

Related work. The question of testing the planarity of partially drawn graphs fits into the general paradigm of extending a partial solution to a full one. This has been studied in various settings and often the extendability problem is more difficult than the unconstrained one. As an example, graph coloring is NP-complete for perfect graphs even if only four vertices are already colored [KS97], while the chromatic number of a perfect graph can be determined in polynomial time [GLS88]. Another example is provided by edge colorings – deciding 3-edge-colorability of cubic bipartite graphs if some edges are already colored is NP-complete [Fia03], while it follows from the famous Kőnig-Hall theorem that cubic bipartite graphs are always 3-edge colorable. In view of these hardness results it is somewhat surprising that the planarity of partially drawn graphs can be tested in polynomial time, in fact in linear time, as we show in this chapter. This is all the more so since this problem is known to be NP-hard for drawings where edges are constrained to be straight-line segments [Pat06].

Specific constraints on planar graph drawings have been studied by several authors. See, for example, [TDB88, Tam98, Dor02, GKM08]. Unfortunately, none of those results can be exploited to solve the question we pose in this chapter. Mohar [Moh99, JM05] gives algorithms for extending 2-cell embeddings on the torus and surfaces of higher genus. However, the 2-cell embedding is a very strong condition that substantially changes the nature of the problem.

Contribution and Outline. In order to solve the general problem, we allow disconnected or low connected graphs to be part of the input. It is readily seen that in this case the rotation schemes (that is, the cyclic orderings of the edges incident to the vertices of the graph) do not fully describe the input. In fact, the relative position of vertices against cycles in the graph must also be considered. (These concepts and their technical details are discussed later.) Further, we make use of the fact that drawing graphs on the plane and on the sphere are equivalent concepts. The advantage of considering embeddings on the sphere lies in the fact that we do not need to distinguish between the outer face and the inner faces.

The main idea of our algorithm is to look at the problem from the "opposite" perspective. Namely, we do not try to directly extend the input partial embedding (which seems much harder than one would expect). Instead, we look at the possible embeddings of the entire graph and decide if any of them contains the partially embedded part as prescribed by the input.

Our algorithm is based on several combinatorial lemmas, relating the problem to the connectivity of the graph. Most of them exhibit the "oncas" property – the obvious necessary conditions are also sufficient. This is particularly elegant in the case of 2-connected graphs. In this case, we exploit the SPQR-tree decomposition of the graph. This notion was introduced in by Di Battista and Tamassia [DT96a] to describe all the possible embeddings of 2-connected planar graphs in a succinct way and was used in various situations when asking for planar embeddings with special properties. A survey on the use of this technique in planar graphs is given by Mutzel [Mut03]. It is indeed obvious that if a 2-connected graph admits a feasible drawing, then the skeleton of each node of the SPQR-tree has a drawing *compatible* (a precise definition of compatibility will come later) with the partial embedding. We prove that the converse is also true. Hence – if we only aim at polynomial running time – we do not need to perform *any* dynamic programming on the SPQR-tree and we could process its nodes independently. However, for the ultimate goal of linear running time, we must refine the approach and pass several pieces of information through the SPQR-tree. Then, dynamic programming becomes more than useful. Also, the

SPQR-trees are exploited at two levels of abstraction, both for decomposing an entire block and for computing the embedding of the subgraph induced by each face of the constrained part of the drawing.

This chapter is organized as follows. We first describe the terminology and list auxiliary topological lemmas, see Section 6.2. In particular, the combinatorial invariants of equivalent embeddings are introduced. In Section 6.3 we state the combinatorial characterization theorems for 2-connected, connected, and disconnected cases. The consequence of them is a simple polynomial-time algorithm outlined at the end of the section. Section 6.4 is then devoted to describe technical details of the linear-time algorithm. Section 6.5 discusses several possible generalizations of the question leading to NP-hard problems, and shows how our techniques can be used to solve other graph drawing problems. We summarize our results and discuss some directions for further research in Section 6.6.

6.2. Notation and Preliminaries

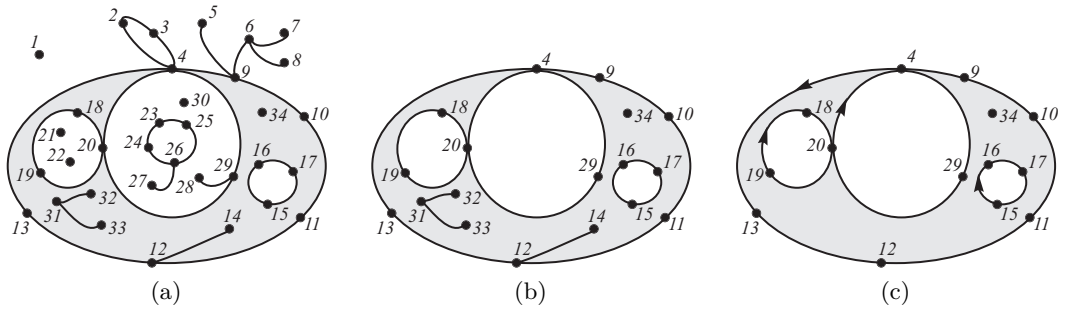
In this section we introduce some notations and preliminaries that are specific to this chapter. In particular, we give a detailed description how planar embeddings of not necessarily connected graphs can be handled, and we give a first characterization of feasible embeddings extensions. We conclude with an overview of data structures and their efficient construction, which will be particularly important for the linear-time implementation of our algorithm.

6.2.1. Drawings, Embeddings, and the Problem Definition

Recall that a planar embedding is an equivalence class of planar drawings. For connected graphs in the plane such an equivalence class can be completely described by the rotation scheme, that is the circular ordering of the edges incident to each vertex, and the external face. For not necessarily connected graphs this is not sufficient, as it does not cover the relative positions of the connected components. This additional information is encoded in the face boundaries, which for each face consists of a list of circular lists of vertices. They are constructed by visiting the (not necessarily connected) border of a face f in such a way to keep f to the left. Together with the rotation scheme and the information which face is the external face, they completely describe planar embeddings, even for non-connected graphs; see Figure 6.1a and 6.1b for an example.

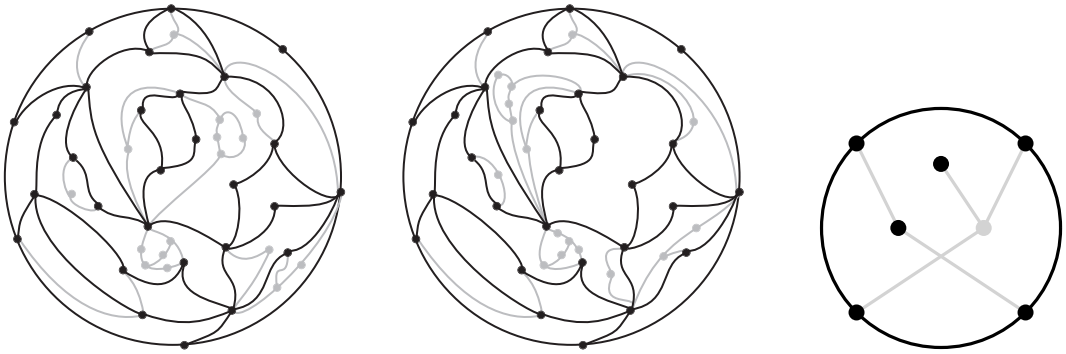
While our initial motivation was to extend a given drawing of a subgraph of a planar graph to a planar drawing of the complete graph, it is not hard to see that this is equivalent to an embedding problem, where we wish to extend a planar embedding of a subgraph to a planar embedding of the whole graph.

A *partially embedded graph*, or PEG for short, is a triplet $(G, H, \mathcal{H})$ where G is a graph, H is a subgraph of G, and $\mathcal{H}$ is a planar embedding of H. We say that the vertices and edges of H are *prescribed*. The problem PARTIALLYEMBEDDEDPLANARITY (PEP) asks whether a given PEG $(G, H, \mathcal{H})$ admits a planar (non-crossing) embedding $\mathcal{G}$ of G whose restriction to H is $\mathcal{H}$. In this case we say that the PEG $(G, H, \mathcal{H})$ is *planar*. We say that $\mathcal{G}$ is an *extension* of an embedding $\mathcal{H}$ of H if the restriction of $\mathcal{G}$ to H is $\mathcal{H}$. See Figure 6.2 for an example of a PEG that admits two different embedding extension and an example that does not admit one.



(a) (b) (c)

Figure 6.1.: (a) A planar drawing of a graph G. The shaded region represents a face f of the drawing. (b) The boundary of f. The circular lists defining the boundary of f are: $[15, 16, 17], [33, 31, 32, 31], [13, 12, 14, 12, 11, 10, 9, 4, 29, 20, 19, 18, 20, 4]$. (c) The facial cycles of f.



(a) (b) (c)

Figure 6.2.: Two different planar embeddings of a graph G whose restrictions to H (black vertices and edges) coincide with $\mathcal{H}$ (a), (b). An instance that does not admit an embedding extension (c). Vertices and edges in $G \setminus H$ are grey.

6.2.2. Facial Cycles and H-Bridges

Let Γ be a planar drawing of a graph H; see Figure 6.1a. Let $\vec{C}$ be a simple cycle in H with an arbitrary orientation. The oriented cycle $\vec{C}$ splits the plane into two connected parts. Denote by $V_\Gamma^{\text{left}}(\vec{C})$ and $V_\Gamma^{\text{right}}(\vec{C})$ the sets of vertices of the graph that are to the left and to the right of $\vec{C}$ in Γ, respectively. The boundary of each face f of Γ can be uniquely decomposed into simple edge-disjoint cycles, bridges (that is, edges that are not part of a cycle), and isolated vertices; see Figure 6.1b. Orient the cycles in such a way that f is to the left when walking along the cycle according to the orientation. Call these oriented cycles the *facial cycles* of f; see Figure 6.1c. Observe that the sets $V_\Gamma^{\text{left}}(\vec{C})$, $V_\Gamma^{\text{right}}(\vec{C})$ and the notion of facial cycles only depend on the embedding $\mathcal{H}$ of Γ. Hence, it makes sense to denote $V_\mathcal{H}^{\text{left}}(\vec{C})$ and $V_\mathcal{H}^{\text{right}}(\vec{C})$, and to consider the facial cycles of $\mathcal{H}$.

Let x be a vertex of a graph G with embedding $\mathcal{G}$. Denote by $E_G(x)$ the set of edges incident to x and by $\sigma_\mathcal{G}(x)$ the rotation scheme of x in $\mathcal{G}$. The following lemma characterizes the planar embeddings $\mathcal{G}$ of a PEG $(G, H, \mathcal{H})$ that extend $\mathcal{H}$ in terms of the rotation scheme and relative cycle–vertex positions with respect to the facial cycles of $\mathcal{H}$.

Lemma 6.1. *Let $(G, H, \mathcal{H})$ be a* PEG *and let $\mathcal{G}$ be a planar embedding of G. The restriction of $\mathcal{G}$ to H is $\mathcal{H}$ if and only if the following conditions hold:*

1) for every vertex $x \in V(H)$, $\sigma_{\mathcal{G}}(x)$ restricted to $E_H(x)$ coincides with $\sigma_{\mathcal{H}}(x)$, and

2) for every facial cycle $\vec{C}$ of each face of $\mathcal{H}$, we have that $V^{\text{left}}_{\mathcal{H}}(\vec{C}) \subseteq V^{\text{left}}_{\mathcal{G}}(\vec{C})$ and $V^{\text{right}}_{\mathcal{H}}(\vec{C}) \subseteq V^{\text{right}}_{\mathcal{G}}(\vec{C})$.

Proof. The proof easily descends from the following statement. Let Γ_1 and Γ_2 be two drawings of the same graph G such that, for every vertex $x \in V(G)$, $\sigma_{\Gamma_1}(x) = \sigma_{\Gamma_2}(x)$. Drawings Γ_1 and Γ_2 have the same embedding if and only if Γ_1 and Γ_2 have the same oriented facial cycles and for each facial cycle $\vec{C}$ we have $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$.

We need to prove this statement in both directions: (i) if Γ_1 and Γ_2 have the same embedding then they have the same oriented facial cycles and for each facial cycle we have $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$ and (ii) if Γ_1 and Γ_2 have the same oriented facial cycles and for each facial cycle we have $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$, then Γ_1 and Γ_2 have the same embedding.

The first direction trivially descends from the observation that drawings with the same embedding have the same facial cycles. Suppose for a contradiction that, for some facial cycle $\vec{C}$, $V^{\text{left}}_{\Gamma_1}(\vec{C}) \neq V^{\text{left}}_{\Gamma_2}(\vec{C})$. Then, at least one vertex v is to the left of $\vec{C}$ in Γ_1 and to the right of $\vec{C}$ in Γ_2 (the opposite case being analogous). Hence, v is part of the boundary of a face that is to the left of $\vec{C}$ in Γ_1 and to the right of $\vec{C}$ in Γ_2, contradicting the hypothesis that Γ_1 and Γ_2 have the same facial boundaries.

For the second direction, first suppose that G is connected and has at least one vertex of degree three. In this case, the fact that Γ_1 and Γ_2 have the same rotation scheme implies that they also have the same face boundaries, and, hence, the same embedding. Second, suppose that G is connected and has maximum degree two. Then, G is either a path or a cycle. In both cases, the face boundaries of Γ_1 and Γ_2 are the same (recall that G is drawn on the sphere). Finally, suppose that G has several connected components C_1, C_2, ..., C_k. Then, Γ_1 and Γ_2 have the same face boundaries if: (a) for each C_i, $i = 1, \ldots, k$, the embedding $\mathcal{G}_1$ of Γ_1 restricted to C_i is the same as the embedding $\mathcal{G}_2$ of Γ_2 restricted to C_i and (b) each pair of connected components C_i and C_j, with $i, j = 1, \ldots, k$ and $i \neq j$, either do not share a face both in $\mathcal{G}_1$ and in $\mathcal{G}_2$ or they contribute with the same circular lists to the boundary of the same face f in $\mathcal{G}_1$ and in $\mathcal{G}_2$.

Condition (a) is guaranteed as in the two cases in which G is connected. Condition (b) follows from the hypothesis that, for each facial cycle $\vec{C}$, we have $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$. In fact, suppose for a contradiction that two connected components C_x and C_y share a face f in $\mathcal{G}_1$ and no face in $\mathcal{G}_2$. Since C_x and C_y share a face in $\mathcal{G}_1$, they are on the same side of any facial cycle $\vec{C}$ belonging to any other component C_z (more intuitively, C_x and C_y can not be separated by any facial cycle of Γ_1). On the other hand, consider the unique path $C_x, f_1, C_1, f_2, \ldots, C_y$ in the component–face tree of $\mathcal{G}_2$. By hypothesis, $C_1 \neq C_x, C_y$. Hence, the facial cycle $\vec{C}$ obtained from the boundary of f_1 and containing vertices of C_1 separates C_x from C_y, thus contradicting the hypothesis that $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$.

Finally, suppose for a contradiction that two connected components C_x and C_y contribute with circular lists L^x_1 and L^y_1 to the boundary of the same face f_1 of $\mathcal{G}_1$ and with circular lists L^x_2 and L^y_2 to the boundary of the same face f_2 of $\mathcal{G}_2$ and suppose that $L^x_1 \neq L^x_2$. The boundary of f_1 is oriented in such a way that every facial cycle has f_1 to its left. Then, every facial cycle obtained from L^x_1 has C_y to its left. Further, for every cycle C' of C_x that is not a facial cycle obtained from L^x_1, there exists a facial cycle $\vec{C}$ obtained from L^x_1 that has C' to its right (part of $\vec{C}$ and of C' may coincide). As $\mathcal{G}_1$ and $\mathcal{G}_2$ restricted to C_x give the same embedding, the last statement is true both in $\mathcal{G}_1$ and in $\mathcal{G}_2$. Then, for every facial cycle $\vec{C}'$ obtained from L^x_2 and not from L^x_1, there exists a facial cycle $\vec{C}$ obtained from L^x_1

hat has $\vec{C}'$ to its right. Since $\vec{C}'$ is incident to f_2 and since C_y is incident to f_2, such a component is to the right of $\vec{C}$, contradicting the hypothesis that $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$. □

Let G be a graph and let H be a subgraph of G. An *H-bridge* K of G is a subgraph of G formed either by a single edge $e \in E(G) \setminus E(H)$ whose end-vertices belong to H or by a connected component K^- of $G - V(H)$, together with all the edges (and their end-vertices) that connect a vertex in K^- to a vertex in H. In the first case, the H-bridge is *trivial*. A vertex that belongs to $V(H) \cap V(K)$ is called an *attachment vertex* (or *attachment*) of K. Note that the edge-sets of the H-bridges form a partition of $E(G) \setminus E(H)$.

An H-bridge K is *local* to a block B of H if all the attachments of K belong to B. Notice that an H-bridge with a single attachment can be local to more than one block, while an H-bridge with at least two attachments is local to at most one block. An H-bridge that is not local to any block of H is *non-local*.

6.2.3. Connectivity and Data Structures

Let $(G, H, \mathcal{H})$ be a PEG. In the following we define some data structures that are widely used throughout the chapter. All of these data structures can be easily computed in time linear in the number of edges of the graph or of the embedding they refer to.

We use the decomposition of a graph G into its connected, biconnected, and triconnected components. To further connect these decompositions with the embedding of $\mathcal{H}$ we make use of several auxiliary data structures.

The *component–face tree* $\mathcal{CF}$ of $\mathcal{H}$ is a tree whose nodes are the connected components of H and the faces of $\mathcal{H}$. A face f and a component C are joined by an edge if a vertex of C is incident to f. The *block–face tree* $\mathcal{BF}$ of $\mathcal{H}$ is a tree whose nodes are the blocks of H and the faces of $\mathcal{H}$. A face f and a block B are joined by an edge if B contains an edge incident to f. The *vertex–face incidence graph* $\mathcal{VF}$ of $\mathcal{H}$ is a graph whose nodes are the vertices of H and the faces of $\mathcal{H}$. A vertex v and a face f are joined by an edge if v appears on the boundary of f.

For the decomposition into biconnected components we use the block–cutvertex tree, as defined in Chapter 2. The *enriched block–cutvertex tree* of a connected graph G is a tree obtained by adding to the block-cutvertex tree of G each vertex v of G that is not a cutvertex and by connecting v to the unique block it belongs to.

To handle the decomposition of a graph into its triconnected components we use the SPQR-tree $\mathcal{T}$ of G, which describes the arrangement of its triconnected components, see Chapter 2.3 for a precise definition. The SPQR-tree is well-suited for the implementation of efficient algorithms, as the SPQR-tree of a biconnected graph on n vertices has $O(n)$ Q-, S-, P-, and R-nodes, the total size of all skeletons of all nodes of $\mathcal{T}$ is also $O(n)$ [BDD00], and the SPQR-tree of G can be computed in linear time [GM00]. We say that an edge e of G *projects* to a virtual edge e' (or *belongs* to e') of $\text{skel}(\mu)$, for some node μ in $\mathcal{T}$, if e belongs to the expansion graph of e'.

Recall that the SPQR-tree $\mathcal{T}$ can be used to represent all the planar embeddings of G. For our characterization we will consider drawings on the sphere, and hence we use the unrooted version of the SPQR-tree. In the second part of the chapter, for the linear-time implementation we will then root the tree to allow for dynamic programming on the tree in a bottom-up fashion.

We emphasize the following properties, that are implicitly exploited throughout all the chapter.

Property 6.1. A planar embedding of the skeleton of every node of $\mathcal{T}$ determines a planar embedding of G and vice versa.

Property 6.2. Let C be a cycle of G and let μ be any node of $\mathcal{T}$. Then, either the edges of C belong to a single virtual edge of $\text{skel}(\mu)$, or they belong to a set of virtual edges that induce a cycle in $\text{skel}(\mu)$.

Efficient computation of data structures. We now briefly discuss the time complexity of constructing the introduced data structures. We further show how to implement the basic queries we use in our algorithms in constant time per operation.

First, observe that a linear-time preprocessing can associate each edge of a planar graph with the unique connected component it belongs to, with the unique block it belongs to, and (given a planar embedding of the graph) with the at most two faces it is incident to. Additionally, we can associate each vertex of a graph with the unique connected component it belongs to.

The block-cutvertex tree of a connected planar graph and the SPQR-tree of a biconnected planar graph can be constructed in linear time [Tar72, GM00]. The enriched block-cutvertex tree of a connected planar graph G can be constructed starting from the block-cutvertex tree of G by adding to the tree (i) each vertex v that is not a cutvertex of G, and (ii) an edge between v and the only block it belongs to.

The block-face tree $\mathcal{BF}$ of a planar embedding $\mathcal{G}$ of a planar graph G can be constructed in linear time. Namely, for each edge e of G, let B_e be the unique block of G containing e and let f'_e and f''_e be the two faces of $\mathcal{G}$ adjacent to e (possibly $f'_e = f''_e$). Add edges (f'_e, B_e) and (f''_e, B_e) to $\mathcal{BF}$. When all the edges of G have been considered, the resulting multigraph $\mathcal{BF}$ has a linear number of edges. Remove multiple edges as follows. Root $\mathcal{BF}$ at any node and orient $\mathcal{BF}$ so that all the edges point toward the root. Remove all the edges, except for one, exiting from each node, thus obtaining the block-face tree $\mathcal{BF}$ of $\mathcal{G}$. The component-face tree $\mathcal{CF}$ of a planar embedding $\mathcal{G}$ of a planar graph G can be constructed in linear time, analogously.

The vertex-face incidence graph $\mathcal{VF}$ of a planar embedding $\mathcal{G}$ of a planar graph G can be constructed in linear time by processing faces of $\mathcal{G}$ one by one, where for each face f we walk along the boundary of f and add to $\mathcal{VF}$ edges between f and the vertices on the boundary. To avoid adding multiple edges, we remember, for each vertex x, the last face f that has been connected to x in $\mathcal{VF}$. Note that $\mathcal{VF}$ is again planar.

Kowalik and Kurowski [KK03] have shown that for a given planar graph F and an integer k, it is possible to build in linear time a "short-path" data structure that allows to check in constant time whether two given vertices of F are connected by a path of length at most k, and return such a path if it exists. We will employ this data structure to search for paths of lengths 1 and 2 in our auxiliary graphs. Using this data structure, we can, for example, determine in constant time whether two vertices share a common face in $\mathcal{H}$ (by finding a path of length two in the vertex-face incidence graph $\mathcal{VF}$) or whether they share the same block (by finding a path of length 2 in the enriched block-cutvertex tree).

6.3. Combinatorial Characterization

We first present a combinatorial characterization of the partially embedded graphs that allow an embedding extension. This not only forms a basis of our algorithm, but it is also interesting in its own right, since it shows that a PEG has an embedding extension if

and only if it satisfies simple conditions that are obviously necessary for an embedding extension to exist.

Our characterization is based on a decomposition of the graph G of a PEG $(G, H, \mathcal{H})$ into its connected, biconnected and triconnected components. For triconnected PEGs the problem is particularly easy. For a triconnected PEG $(G, H, \mathcal{H})$ the graph G has only two distinct planar embeddings $\mathcal{G}_1$ and $\mathcal{G}_2$. The PEG is thus planar if and only if either $\mathcal{G}_1$ or $\mathcal{G}_2$ extends $\mathcal{H}$. Clearly, for a non-connected PEG in order to admit an embedding extension it is as necessary condition that each of its connected components admits an embedding extension. Similarly, it is a necessary condition for a connected PEG that at least each biconnected component admits an embedding extension. We start with the most specific case, the case where G is biconnected and then extend the characterization from there to the cases where G is connected or even disconnected.

6.3.1. Planarity of Biconnected Pegs

In this section we focus on biconnected PEGs $(G, H, \mathcal{H})$, that is the graph G is biconnected. This assumption allows us to use the SPQR-tree of G as the main tool of our characterization. The characterization is based on the two necessary and sufficient conditions of Lemma 6.1. We show that they can be individually translated to constraints on the embeddings of the skeletons of the SPQR-tree of G.

Definition 6.1. A planar embedding of the skeleton of a node of the SPQR-tree of G is *edge-compatible with* $\mathcal{H}$ if, for every vertex x of the skeleton and for every three edges of $E_H(x)$ belonging to different virtual edges of the skeleton, their clockwise order determined by the embedding of the skeleton is a suborder of $\sigma_{\mathcal{H}}(x)$.

Lemma 6.2. *Let $(G, H, \mathcal{H})$ be a biconnected* PEG*. Let $\mathcal{T}$ be the SPQR-tree of G. An embedding $\mathcal{G}$ of G satisfies condition 1 of Lemma 6.1 if and only if, for each node μ of $\mathcal{T}$, the corresponding embedding of* skel(μ) *is edge-compatible with $\mathcal{H}$.*

Proof. Obviously, if G has an embedding satisfying condition 1 of Lemma 6.1, then the corresponding embedding of skel(μ) is edge-compatible with $\mathcal{H}$, for each node μ of $\mathcal{T}$.

To prove the converse, assume that the skeleton of every node of $\mathcal{T}$ has an embedding that is edge-compatible with $\mathcal{H}$, and let $\mathcal{G}$ be the embedding of G determined by all such skeleton embeddings. We claim that $\mathcal{G}$ satisfies condition 1 of Lemma 6.1. To prove the claim, it suffices to show that any three edges e, f, and g of $\mathcal{H}$ that share a common vertex x appear in the same clockwise order around x in $\mathcal{H}$ and in $\mathcal{G}$. Assume that the triple (e, f, g) is embedded in clockwise order around x in $\mathcal{H}$. Let μ be the node of $\mathcal{T}$ with the property that the Q-nodes representing e, f, and g appear in distinct components of $\mathcal{T} - \mu$. Note that such a node μ exists and is unique. The three edges e, f, and g project into three distinct virtual edges e', f', and g' of skel(μ). Since the embedding of skel(μ) is assumed to be edge-compatible with $\mathcal{H}$, the triple (e', f', g') is embedded in clockwise order in skel(μ), and hence the triple (e, f, g) is embedded in clockwise order in $\mathcal{G}$. □

This settles the translation of condition 1 of Lemma 6.1 to conditions on the embeddings of the skeletons of the SPQR-tree of G. Next, we deal with condition 2. Consider a simple cycle $\vec{C}$ of G with an arbitrary orientation and a node μ of the SPQR-tree of G. Either all the edges of $\vec{C}$ belong to the expansion graph of a single virtual edge of skel(μ) or the virtual edges whose expansion graphs contain the edges of $\vec{C}$ form a simple cycle in skel(μ). Such a cycle in skel(μ) inherits the orientation of $\vec{C}$ in a natural way.

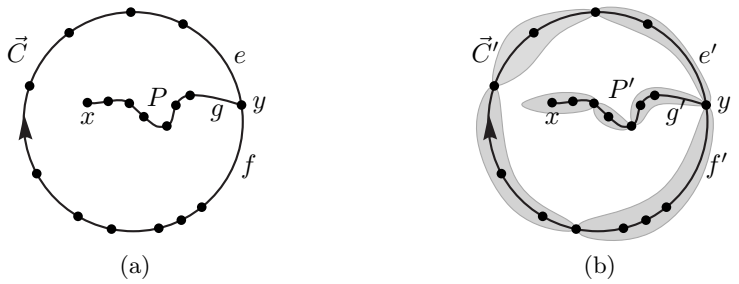


Figure 6.3.: Illustration for the proof of Lemma 6.3. Grey regions represent virtual edges of the skeleton of a node of $\mathcal{T}$.

Definition 6.2. A planar embedding of the skeleton of a node μ of the SPQR-tree of G is *cycle-compatible with* $\mathcal{H}$ if, for every facial cycle $\vec{C}$ of $\mathcal{H}$ whose edges project to a simple cycle $\vec{C}'$ in skel(μ), all the vertices of skel(μ) that belong to $V_{\mathcal{H}}^{\text{left}}(\vec{C})$ and all the virtual edges that contain vertices of $V_{\mathcal{H}}^{\text{left}}(\vec{C})$ (except for the virtual edges of $\vec{C}'$ itself) are embedded to the left of $\vec{C}'$; and analogously for $V_{\mathcal{H}}^{\text{right}}(\vec{C})$.

Lemma 6.3. *Let $(G, H, \mathcal{H})$ be an instance of* Pep *where G is biconnected. Let $\mathcal{T}$ be the SPQR-tree of G. An embedding $\mathcal{G}$ of G satisfies condition 2 of Lemma 6.1 if and only if, for each node μ of $\mathcal{T}$, the corresponding embedding of* skel(μ) *is cycle-compatible with $\mathcal{H}$.*

Proof. Obviously, if $\mathcal{G}$ is an embedding of G that satisfies condition 2 of Lemma 6.1, then the corresponding embedding of skel(μ) is cycle-compatible with $\mathcal{H}$, for each node μ of $\mathcal{T}$.

To prove the converse, assume that skel(μ) has an embedding that is cycle-compatible with $\mathcal{H}$, for each node μ of $\mathcal{T}$, and let $\mathcal{G}$ be the resulting embedding of G.

Our goal is to show that, for every facial cycle $\vec{C}$ of $\mathcal{H}$ and for every vertex x of $H - V(\vec{C})$, the relative left/right position of x with respect to $\vec{C}$ is the same in $\mathcal{H}$ as in $\mathcal{G}$.

Refer to Fig. 6.3. Assume that x is to the right of $\vec{C}$ in $\mathcal{G}$ (the other case being analogous). Let P be the shortest path in G that connects x to a vertex of $\vec{C}$. Such a path exists since G is connected. Let y be the vertex of $\vec{C} \cap P$, and let e and f be the two edges of $\vec{C}$ adjacent to y, where e directly precedes f in the orientation of $\vec{C}$. By the minimality of P, all the vertices of $P - y$ avoid $\vec{C}$, hence all the vertices of $P - y$ are to the right of $\vec{C}$ in $\mathcal{G}$. Let g be the edge of P adjacent to y. In $\mathcal{G}$, the triple (e, f, g) appears in clockwise order around y.

Let μ be the (unique) internal node of $\mathcal{T}$ in which e, f, and g project to distinct edges e', f', and g' of skel(μ). Let $\vec{C}'$ be the projection of $\vec{C}$ into skel(μ) (in other words, $\vec{C}'$ is the subgraph of skel(μ) formed by edges that contain the projection of at least one edge of $\vec{C}$), and let P' be the projection of P. It is easy to see that $\vec{C}'$ is a cycle of length at least two, while P' is either a path or a cycle. Assume that the edges of $\vec{C}'$ are oriented consistently with the orientation of $\vec{C}$ and that the edges of P' form an ordered sequence, where the edge containing x is the first and g' is the last.

Both the endpoints of an edge of $\vec{C}'$ are vertices of $\vec{C}$. Analogously, both the endpoints of an edge of P' are vertices of P, with the possible exception of the first vertex of P'. It follows that no vertex of P' belongs to $\vec{C}'$, except possibly for the first one and the last one. Thus, no edge of P' belongs to $\vec{C}'$ and, by the assumption that the embedding of skel(μ) is planar and that $\mathcal{G}$ is the embedding resulting from the skeleton embedding choices, all the edges of P' are embedded to the right of the directed cycle $\vec{C}'$ in skel(μ). In particular,

the edge of skel(μ) containing x is to the right of $\vec{C}'$. Since the embedding of skel(μ) is assumed to be cycle-compatible with $\mathcal{H}$, x is to the right of $\vec{C}$ in $\mathcal{H}$.

This shows that $\mathcal{G}$ satisfies condition 2 of Lemma 6.1, as claimed. □

Definition 6.3. A planar embedding of the skeleton of a node μ of the SPQR-tree of G is *compatible with* $\mathcal{H}$ if it is both edge- and cycle-compatible with $\mathcal{H}$.

As a consequence of Lemmas 6.2 and 6.3, we obtain the following result, characterizing the positive instances of Pep in which G is biconnected.

Theorem 6.1. *Let $(G, H, \mathcal{H})$ be an instance of* Pep *where G is biconnected. Then G has an embedding which extends $\mathcal{H}$ if and only if the skeleton of each node of its SPQR-tree has an embedding compatible with $\mathcal{H}$.*

If G is biconnected we can use Theorem 6.1 for devising a polynomial-time algorithm for Pep. Namely, we can test, for each node μ of the SPQR-tree $\mathcal{T}$ of G whether skel(μ) has an embedding that is compatible with $\mathcal{H}$. For Q-, S-, and R-nodes, this test is easily done in polynomial time.

If μ is a P-node, the test is more complex. Let x and y be the two poles of skel(μ). We say that a virtual edge e of skel(μ) is *constrained* if the expansion graph of e (that is, the pertinent graph of the child node of μ in $\mathcal{T}$ corresponding to e) contains at least one edge of H incident to x and at least one edge of H incident to y. To obtain an embedding of μ edge-compatible with $\mathcal{H}$, the constrained edges must be embedded in a cyclic order that is consistent with $\sigma_{\mathcal{H}}(x)$ and $\sigma_{\mathcal{H}}(y)$. Such a cyclic order, if it exists, is unique and can be determined in polynomial time. Note that, if $\mathcal{H}$ has a facial cycle $\vec{C}$ that projects to a proper cycle $\vec{C}'$ in μ, then $\vec{C}'$ has exactly two edges and these two edges are both constrained. Thus, the embedding of any such cycle $\vec{C}'$ in μ is fixed as soon as we fix the cyclic order of the constrained edges. Once the cyclic order of the constrained edges of μ is determined, we process the remaining edges one-by-one and insert them among the edges that are already embedded, in such a way that no edge-compatibility or cycle-compatibility constraints are violated. It is not difficult to verify that this procedure constructs an embedding of μ compatible with $\mathcal{H}$, if such an embedding exists.

Thus, Pep can be solved in polynomial time for biconnected Pegs.

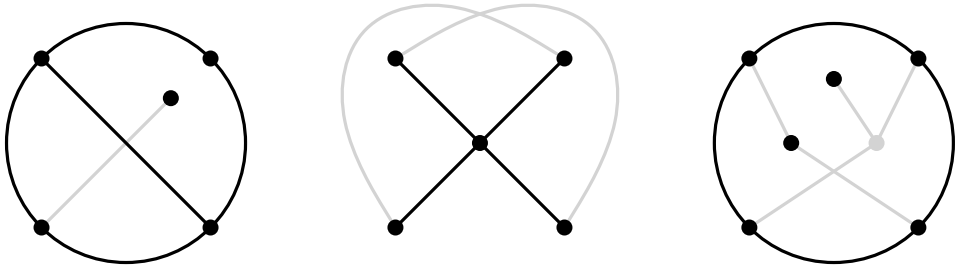
6.3.2. Planarity of Connected and Disconnected Pegs

A graph is planar if and only if each of its blocks is planar. Thus, planarity testing of general graphs can be reduced to planarity testing of biconnected graphs. For planarity testing of partially embedded graphs, the same simple reduction does not work; see Figure 6.4. However, we will show that solving partially embedded planarity for a general instance $(G, H, \mathcal{H})$ can be reduced to solving the subinstances induced by the blocks of G and to checking additional conditions that guarantee that the partial solutions can be combined into a full solution for G.

Let us consider connected Pegs $(G, H, \mathcal{H})$, that is instances of Pep, in which G is connected. When dealing with such an instance, it is useful to assume that G has no non-trivial non-local H-bridge. We will now show that any instance of Pep can be transformed to an equivalent instance that satisfies this additional assumption.

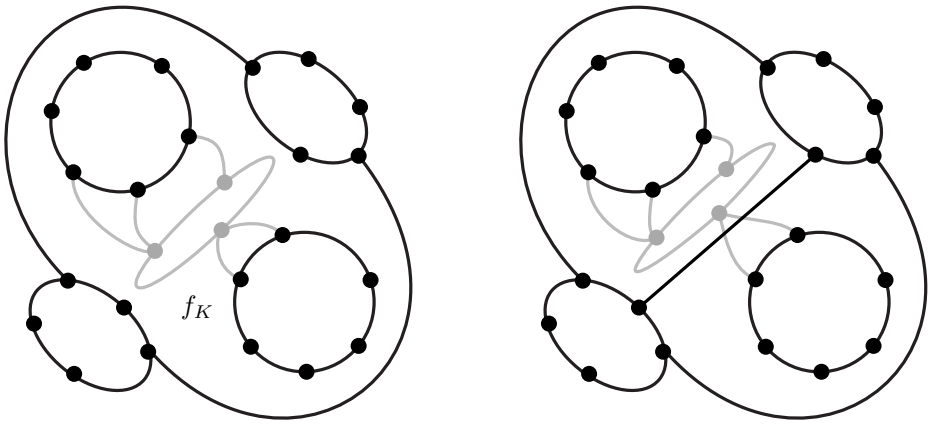
Let K be a non-trivial non-local H-bridge of G. Since K is non-local, it must have at least two attachments, and these attachments do not belong to any single block of H.

Let f_K be the face of $\mathcal{H}$ whose boundary contains all the attachments of the H-bridge K. Note that there can be at most one such a face (see Fig. 6.5.a): If the attachments of K



(a) (b) (c)

Figure 6.4.: Three examples of PEP instances $(G, H, \mathcal{H})$ that have no embedding extension, even though each block of G admits an embedding extending the corresponding sub-embedding of $\mathcal{H}$. The black edges and vertices represent $\mathcal{H}$, the gray edges and vertices belong to G but not to H. Note that instance (a) fails to satisfy condition 3 of Lemma 6.4, instance (b) fails to satisfy condition 2 of Lemma 6.4, and instance (c) has a non-trivial non-local H-bridge. The modification of instance (c) into an equivalent instance without non-trivial non-local H-bridges creates a block of G that does not have an embedding extension.



(a) (b)

Figure 6.5.: A non-local bridge is either necessarily contained in a face f_K (a) or causes a non-planarity (b).

were contained in the intersection of the boundaries of two distinct faces of $\mathcal{H}$, then K would necessarily be local. If there is no face of $\mathcal{H}$ incident to all the attachments of K, then G clearly has no embedding extension (see Fig. 6.5.b). In this case, we define f_K as an arbitrary face of $\mathcal{H}$.

Let $\mathcal{K}$ be the set of non-trivial non-local H-bridges of G. It is clear that, in any embedding of G extending $\mathcal{H}$, all the vertices of $K - V(H)$ are embedded inside f_K, for every $K \in \mathcal{K}$. This motivates the following definition.

Definition 6.4. Let H' be the graph whose edge set is equal to the edge set of H, and whose vertex set is defined by $V(H') = V(H) \cup \bigcup_{K \in \mathcal{K}} V(K)$. Let $\mathcal{H}'$ be the embedding of H' that is obtained from $\mathcal{H}$ by inserting, for every H-bridge $K \in \mathcal{K}$, all the vertices of $K - V(H)$ into the interior of the face f_K.

Observe that the graph G has no non-trivial non-local H'-bridges. Observe also, that any embedding of G that extends $\mathcal{H}$ also extends $\mathcal{H}'$, and vice versa. Thus, the instance

$(G, H, \mathcal{H})$ of Pep is equivalent to the instance $(G, H', \mathcal{H}')$, which contains no non-trivial non-local bridges.

Before we state the next lemma, we need more terminology. Let $\mathcal{H}$ be an embedding of a graph H, and let H_1 and H_2 be edge-disjoint subgraphs of H. We say that H_1 and H_2 *alternate* around a vertex x of $\mathcal{H}$ if there are two pairs of edges $e, e' \in E(H_1)$ and $f, f' \in E(H_2)$ that are incident to x and that appear in the cyclic order (e, f, e', f') in the rotation scheme of x restricted to these four edges. Let x and y be two vertices of H and let $\vec{C}$ be a directed cycle in $\mathcal{H}$. We say that $\vec{C}$ *separates* x and y if $x \in V_{\mathcal{H}}^{\text{left}}(\vec{C})$ and $y \in V_{\mathcal{H}}^{\text{right}}(\vec{C})$, or vice versa.

Lemma 6.4. *Let $(G, H, \mathcal{H})$ be an instance of Pep where G is connected and every non-trivial H-bridge of G is local. Let $G_1, \ldots, G_t$ be the blocks of G, let H_i be the subgraph of H induced by the vertices of G_i, and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Then, G has an embedding extending $\mathcal{H}$ if and only if*

1) each G_i has an embedding extending $\mathcal{H}_i$,

2) no two distinct graphs H_i and H_j alternate around any vertex of $\mathcal{H}$, and

3) for every facial cycle $\vec{C}$ of $\mathcal{H}$ and for any two vertices x and y of $\mathcal{H}$ separated by $\vec{C}$, any path in G connecting x and y contains a vertex of $\vec{C}$.

Proof. Clearly, the three conditions of the lemma are necessary. To show that they are also sufficient, assume that the three conditions are satisfied and proceed by induction on the number t of blocks of G.

If $t = 1$, then G is biconnected, and there is nothing to prove. Assume that $t \geq 2$. If there is at least one block G_i that does not contain any vertex of H, we restrict our attention to the subgraph G' of G induced by those blocks that contain at least one vertex of H. Since every non-trivial H-bridge of G is local, graph G' is connected, and hence it satisfies the three conditions of the lemma. By induction, the embedding $\mathcal{H}$ can be extended into an embedding $\mathcal{G}'$ of G'. Since every block G_i of G is planar (by condition 1 of the lemma), it is easy to extend the embedding $\mathcal{G}'$ into an embedding of G.

Assume now that every block of G contains at least one vertex of H. This implies that every cutvertex of G belongs to H, because otherwise the cutvertex would belong to a non-local H-bridge, which is impossible by assumption. Let x be any cutvertex of G. Let $G'_1, G'_2, \ldots, G'_k$ be the connected components of $G - x$, where we select G'_1 by the following rules: if there is a component of $G - x$ that has no vertex connected to x by an edge of H, then let G'_1 be such a component; if each component of $G - x$ is connected to x by an edge of H, then choose G'_1 in such a way that the edges of H incident to x and belonging to G'_1 form an interval in $\sigma_{\mathcal{H}}(x)$. Such a choice of G'_1 is always possible, due to condition 2 of the lemma.

Let G' be the subgraph of G induced by $V(G'_1) \cup \{x\}$, and let G'' be the subgraph of G induced by $V(G'_2) \cup \cdots \cup V(G'_k) \cup \{x\}$. Let H' and H'' be the subgraphs of H induced by the vertices of G' and G'', respectively, and let $\mathcal{H}'$ and $\mathcal{H}''$ be $\mathcal{H}$ restricted to H' and H'', respectively. Both G' and G'' have fewer blocks than G. Also, both the instances $(G', H', \mathcal{H}')$ and $(G'', H'', \mathcal{H}'')$ satisfy the conditions of the lemma. Thus, by induction, there is an embedding $\mathcal{G}'$ of G' that extends $\mathcal{H}'$ and an embedding $\mathcal{G}''$ of G'' that extends $\mathcal{H}''$. Our goal is to combine $\mathcal{G}'$ and $\mathcal{G}''$ into a single embedding of G that extends $\mathcal{H}$. To see that this is possible, we prove two auxiliary claims.

Claim 1. $\mathcal{H}'$ has a face f' whose boundary contains x and, for any facial cycle $\vec{C}$ of f', all the vertices of H'' except for x are in $V_{\mathcal{H}}^{\text{left}}(\vec{C})$, that is, they are 'inside' f'.

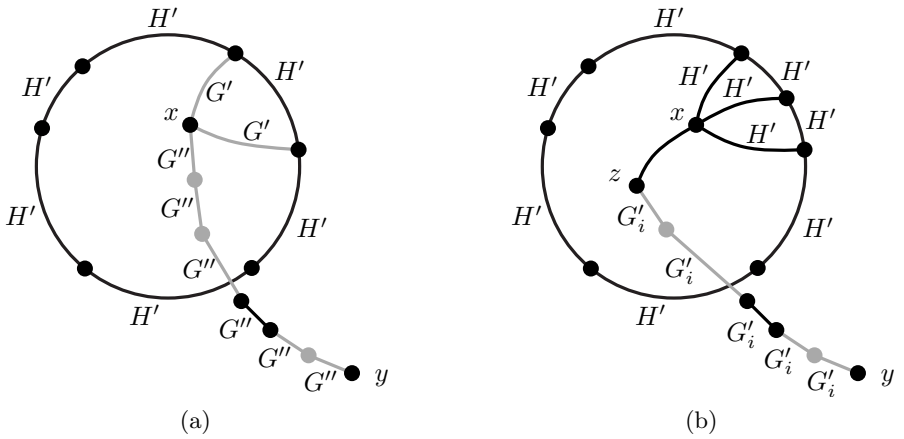


Figure 6.6.: Illustration for the proof of Lemma 6.4. (a) H' has no edge incident to x. (b) H' has an edge incident to x.

To see that the claim holds, assume first that H' has no edge incident to x (see Fig. 6.6.a). Let f' be the unique face of $\mathcal{H}'$ incident to x. We show that all the vertices of H'' are inside f' in $\mathcal{H}$. Let y be any vertex of H''. Since G'' is connected, there is a path P in G'' from y to x. Assume for contradiction that $\mathcal{H}'$ has a facial cycle $\vec{C}$ such that $\vec{C}$ separates y from x in $\mathcal{H}$. This cycle belongs to $H' - x$, hence $\vec{C}$ and P are disjoint, contradicting condition 3 of the lemma.

Next, assume that H' has an edge incident to x (see Fig. 6.6.b). By the construction of G_1, each connected component of $G - x$ has at least one vertex connected to x by an edge of H. Moreover, the edges of $\mathcal{H}'$ incident to x form an interval in $\sigma_{\mathcal{H}}(x)$. This shows that $\mathcal{H}'$ has a face f' containing x on its boundary, and such that every vertex of H'' adjacent to x is inside f' in $\mathcal{H}$. We now show that all the vertices of H'' except for x are inside f'. Let y be a vertex of H'' different from x. Let G'_i be the component of $G - x$ containing y. We know that G'_i has a vertex z adjacent to x by an edge of H and that z is inside f' in $\mathcal{H}$. Let P be a path in G'_i connecting y and z. If y is not inside f', then y is separated from z in $\mathcal{H}$ by a facial cycle of $\mathcal{H}'$, contradicting condition 3 of the lemma.

Claim 2. All the vertices of H', except for x, appear in $\mathcal{H}$ inside the same face f'' of $\mathcal{H}''$; further, x is on the boundary of f''.

To prove the claim, note that any two vertices from $H' - x$ are inside the same face f'' of $\mathcal{H}''$ in $\mathcal{H}$ by condition 3 of the lemma, because they are connected by a path in G'_1. Vertex x is on the boundary of f'', since otherwise it would be separated in $\mathcal{H}$ from the remaining vertices of H' by a facial cycle of f'', again contradicting condition 3 of the lemma.

In view of the previous two claims, it is easy to see that the embedding $\mathcal{G}'$ of G' and the embedding $\mathcal{G}''$ of G'' can be combined into a single embedding $\mathcal{G}$ of G that extends $\mathcal{H}$. To see this, note that, when $\mathcal{H}'$ is extended into $\mathcal{G}'$, the face f' from Claim 1 can be subdivided into several faces of $\mathcal{G}'$, at least one of which, say g', contains x on its boundary. Analogously, the face f'' from Claim 2 can be subdivided into several faces of $\mathcal{G}''$, at least one of which, say g'', contains x on its boundary. We then obtain the embedding $\mathcal{G}$ by merging the faces g' and g'' into a single face. □

Observe that computing the non-local H-bridges $\mathcal{K}$ and for each non-local H-bridge $K \in \mathcal{K}$ the face f_K can be done in polynomial time. Afterwards, the second and third

conditions of Lemma 6.4 can be easily checked in polynomial time.

Next, we focus on disconnected PEGs, that is the instances $(G, H, \mathcal{H})$ of PEP in which G is not connected. The possibility of solving the subinstances of $(G, H, \mathcal{H})$ induced by the connected components of G does not guarantee that the instance $(G, H, \mathcal{H})$ of PEP has a solution. However, we show that solving PEP for an instance $(G, H, \mathcal{H})$ can be reduced to solving the subinstances induced by the connected components of G and to checking additional conditions that guarantee that the partial solutions can be combined into a full solution for G.

Lemma 6.5. *Let $(G, H, \mathcal{H})$ be an instance of* PEP*. Let $G_1, \ldots, G_t$ be the connected components of G. Let H_i be the subgraph of H induced by the vertices of G_i, and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Then G has an embedding extending $\mathcal{H}$ if and only if:*

1) each G_i has an embedding extending $\mathcal{H}_i$, and

2) for each i, for each facial cycle $\vec{C}$ of H_i and for every $j \neq i$, no two vertices of H_j are separated by $\vec{C}$.

Proof. Clearly, the two conditions of the lemma are necessary. To show that they are also sufficient, assume that the two conditions are satisfied and proceed by induction on the number t of connected components of G.

If $t = 1$ then G is connected and there is nothing to prove. Assume now that G has $t \geq 2$ connected components $G_1, \ldots, G_t$. Let H_i and $\mathcal{H}_i$ be defined as in the statement of the lemma. Note that the graphs H_i may consist of several connected components. Let $\mathcal{CF}$ be the component-face tree of $\mathcal{H}$, rooted at a node that represents an arbitrary face of $\mathcal{H}$. We say that a face f_i of $\mathcal{H}$ is the *outer face* of $\mathcal{H}_i$ if at least one child of f_i in $\mathcal{CF}$ is a component of H_i, but the parent of f_i is not a component of H_i. Observe that, due to the second condition of the lemma, each $\mathcal{H}_i$ has exactly one outer face f_i. We thus have a sequence of (not necessarily distinct) outer faces $f_1, \ldots, f_t$ of $G_1, \ldots, G_t$.

Let us now assume, without loss of generality, that in the subtree of $\mathcal{CF}$ rooted at f_1, there is no outer face $f_i \neq f_1$. This implies that f_1 is the only face of $\mathcal{H}$ that is incident both to H_1 and to $H - H_1$. By induction, the embedding $\mathcal{H} - \mathcal{H}_1$ can be extended to an embedding $\mathcal{G}_{\geq 2}$ of the graph $G - G_1$. By the first condition of the lemma, $\mathcal{H}_1$ can be extended into an embedding $\mathcal{G}_1$ of G_1. The two embeddings $\mathcal{H} - \mathcal{H}_1$ and $\mathcal{H}_1$ share a single face f_1.

When extending the embedding $\mathcal{H}_1$ into $\mathcal{G}_1$, the face f_1 of $\mathcal{H}_1$ can be subdivided into several faces of $\mathcal{G}_1$. Let f' be any face of $\mathcal{G}_1$ obtained by subdividing f_1. Analogously, in the embedding $\mathcal{G}_{\geq 2}$ the face f_1 can be subdivided into several faces, among which we choose an arbitrary face f''.

We then glue the two embeddings $\mathcal{G}_1$ and $\mathcal{G}_{\geq 2}$ by identifying the face f' of $\mathcal{G}_1$ and the face f'' of $\mathcal{G}_{\geq 2}$ into a single face whose boundary is the union of the boundaries of f' and f''. This yields an embedding of G that extends $\mathcal{H}$. □

Note that the second condition of Lemma 6.5 can be easily tested in polynomial time. Thus, we can use the characterization to directly prove that PARTIALLYEMBEDDEDPLANARITY is solvable in polynomial time. In the rest of the chapter, we describe a more sophisticated algorithm that solves PEP in linear time.

6.4. Linear-Time Algorithm

In this section we devise a linear-time algorithm for solving Pep. The algorithm basically follows the outline of the characterization. The first milestone is a linear-time algorithm for biconnected Pegs. Afterwards, we show that the additional conditions for connected and disconnected Pegs can be checked in linear time.

Essentially, to solve the biconnected case, it is sufficient to give a linear-time implementation of the algorithm sketched at the end of Section 6.1. In fact, most of the steps sketched there are fairly easy to implement in linear time. The problem of finding a compatible embedding of a P-node, however, is tricky.

One P-node μ may contain a linear number of facial cycles that project to cycles in skel(μ). Further, a linear number of edges may have no H-edge adjacent to the poles of skel(μ). Hence, the positions at which they have to be inserted in the cyclic orderings around the poles of skel(μ) depend on the cycle containment constraints. To process a P-node skel(μ) in time proportional to its size, we would need to find for each virtual edge of P a position where it is contained in exactly the cycles it needs to be in and outside of all other cycles.

Hence, the main problem for reaching linear running time stems from the cycle compatibility constraints, that is condition 2 of Lemma 6.1. The constraints stemming from rotation scheme consist of an ordering of a subset of the incident edges of each vertex. Thus, the total size of these constraints is linear, and, additionally, the constraints are very local. In light of these two properties, it is not surprising that the rotation scheme constraints are relatively simple to handle in linear total time. The same does, however, not hold for the cycle containment constraints. As specified in condition 2 of Lemma 6.1, these constraints specify for each directed facial cycle $\vec{C}$ of H and each vertex v of $H - C$ whether v is to the left or to the right of $\vec{C}$. Note that the graph H may contain a linear number of facial cycles, and thus this amounts to quadratically many cycle–vertex constraints. Further, these constraints do not exhibit any locality on G. Evidently, a lot of the information encoded in the cycle containment constraints is redundant, as the set of cycles involved in these constraints is the set of facial cycles of a planar graph.

We use two different approaches to handle the cycle containment constraints in linear time. One is to ignore them; we prove that this yields a correct solution if H is connected, as in this case the cycle containment constraints are implied by the rotation scheme. The second consists in considering restricted instances, where the constraints can easily be expressed in linear space. Suppose we have a Peg $(G, H, \mathcal{H})$, with a face f of $\mathcal{H}$ such that all vertices of H are part of at least one facial cycle of f. This implies that each facial cycle of H has f on the left side and the right side does not contain any vertices of H. In this case, the cycle containment constraints of each facial cycle $\vec{C}$ of f can be expressed as $V_{\mathcal{H}}^{\text{right}}(\vec{C}) = \emptyset$, thus yielding a set of constraints whose size is linear. Moreover, in the SPQR-tree it is sufficient to keep track which virtual edges contain vertices of H, and which do not. This information is much easier to aggregate than information about individual vertices and all their cycle containment constraints.

First, we tackle the case in which G is biconnected. The algorithm solving this case, presented in Section 6.4.3, uses the algorithms presented in Sections 6.4.1 and 6.4.2 as subroutines to solve more restricted subcases. Then, we deal with the case in which G is singly connected and with the general case, where G may be disconnected, in Subsection 6.4.4. The algorithm we present exploits several auxiliary data structures, namely block-cutvertex trees, SPQR-trees, enriched block-cutvertex trees, block-face trees,

component-face trees, and vertex-face incidence graphs. Note that all these data structures can be easily computed in linear time [GM00].

6.4.1. G Biconnected, H Connected

In this section we show how to solve Pep in linear time for biconnected Pegs $(G, H, \mathcal{H})$ with H connected. We first show that in this case the rotation scheme alone is sufficient for finding an embedding extension.

Lemma 6.6. *Let $(G, H, \mathcal{H})$ be* Peg *such that G is biconnected and H is connected. Let $\mathcal{G}$ be any planar embedding of G satisfying condition 1 of Lemma 6.1. Then, $\mathcal{G}$ satisfies condition 2 of Lemma 6.1.*

Proof. Suppose, for a contradiction, that a planar embedding $\mathcal{G}$ of G exists such that $\mathcal{G}$ satisfies condition 1 and does not satisfy condition 2 of Lemma 6.1. Then, there exists a facial cycle $\vec{C}$ of $\mathcal{H}$ such that either there exists a vertex $x \in V_{\mathcal{H}}^{\text{left}}(\vec{C})$ with $x \in V_{\mathcal{G}}^{\text{right}}(\vec{C})$ or there exists a vertex $x \in V_{\mathcal{H}}^{\text{right}}(\vec{C})$ with $x \in V_{\mathcal{G}}^{\text{left}}(\vec{C})$. Suppose that we are in the former case, as the latter case can be discussed analogously. Since $\mathcal{H}$ is a planar embedding and H is connected, there exists a path $P = (x_1, x_2, \ldots, x_k) \in H$ such that x_1 is a vertex of $\vec{C}$, $x_i \in V_{\mathcal{H}}^{\text{left}}(\vec{C})$, for each $i = 2, \ldots, k$, and $x_k = x$. Denote by x_1^- and by x_1^+ the vertex preceding and following x_1 in the oriented cycle $\vec{C}$, respectively. Consider the placement of x_2 with respect to $\vec{C}$ in $\mathcal{G}$. As $x_2 \notin \vec{C}$, either $x_2 \in V_{\mathcal{G}}^{\text{left}}(\vec{C})$ or $x_2 \in V_{\mathcal{G}}^{\text{right}}(\vec{C})$. In the first case, the path $(x_2, \ldots, x_k)$ crosses $\vec{C}$, since $x_2 \in V_{\mathcal{G}}^{\text{left}}(\vec{C})$, $x_k \in V_{\mathcal{G}}^{\text{right}}(\vec{C})$, and no vertex v_i belongs to $\vec{C}$, for $i = 2, \ldots, k$, thus contradicting the planarity of the embedding $\mathcal{G}$. In the second case, the clockwise order of the edges incident to x_1 in $\mathcal{H}$ is (x_1, x_1^-), (x_1, x_2), and (x_1, x_1^+), while the clockwise order of the edges incident to x_1 in $\mathcal{G}$ is (x_1, x_1^-), (x_1, x_1^+), and (x_1, x_2), thus contradicting the assumption that $\mathcal{G}$ satisfies condition 1 of Lemma 6.1. □

By Lemma 6.6, testing whether a planar embedding $\mathcal{G}$ exists satisfying conditions 1 and 2 of Lemma 6.1 is equivalent to testing whether a planar embedding $\mathcal{G}$ exists satisfying condition 1 of Lemma 6.1. Due to Lemma 6.2, testing whether a planar embedding $\mathcal{G}$ exists satisfying condition 1 is equivalent to testing whether the skeleton of each node of the SPQR-tree of G has a planar embedding that is edge-compatible with $\mathcal{H}$. We now describe an algorithm, called Algorithm BC (for G **B**iconnected and H **C**onnected), that achieves this in linear time.

Algorithm BC. Construct the SPQR-tree $\mathcal{T}$ of G and root it at an arbitrary internal node. This choice determines for each node μ its pertinent graph and also determines one special virtual edge in each skeleton, namely the one that skel(μ) shares with its parent. A bottom-up visit of $\mathcal{T}$ is performed, such that after a node μ of $\mathcal{T}$ has been visited, an embedding of skel(μ) that is edge-compatible with $\mathcal{H}$ is selected, if it exists. For each virtual edge uv of the skeleton skel(μ) of a node μ, in order to find a cycle-compatible embedding, we need to know whether its expansion graph contains H-edges incident to u and v. Due to the bottom-up traversal, all pertinent graphs of all children have already been processed by the algorithm, and we can thus aggregate this information for all edges, except the one the skel(μ) shares with its parent.

In order to keep track of the edges of H that belong to pert(μ) and that are incident to the pole $u(\mu)$, define the *first edge* $f_{u(\mu)}$ and the *last edge* $l_{u(\mu)}$ as the edges of H such that exactly the edges between $f_{u(\mu)}$ and $l_{u(\mu)}$ in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$ belong to pert(μ). The first and last edge of $v(\mu)$ are defined analogously.

After a node μ of $\mathcal{T}$ has been visited by the algorithm, edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ are associated with μ. We can then also refer to them as $f_{u(e)}$ and $l_{u(e)}$ (respectively $f_{v(e)}$ and $l_{v(e)}$) where e is the virtual edge corresponding to μ in the skeleton of the parent of μ.

If μ is a Q- or an S-node, no check is needed. As skel(μ) is a cycle, the only planar embedding of skel(μ) is edge-compatible with $\mathcal{H}$. The edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ are easily computed.

If μ is an R-node, then skel(μ) has only two planar embeddings. For each of them, verify if it is edge-compatible with $\mathcal{H}$ by performing the following check. For each vertex x of skel(μ) restrict the circular list of its incident virtual edges to the virtual edges $e_1, \ldots, e_h$ that contain an edge of H incident to x. Check if $l_{x(e_i)}$ precedes $f_{x(e_{i+1})}$ (for $i = 1, \ldots, h$, where $e_{h+1} = e_1$) in the list of the edges incident to x in $\mathcal{H}$. If x is a pole, do an analogous check on the linear list of its incident virtual edges obtained by removing the virtual edge corresponding to the parent of μ from the circular list. If one of the tests succeeds, then select the corresponding embedding for skel(μ). Set $f_{u(\mu)} = f_{u(f_1)}$, $l_{u(\mu)} = l_{u(f_p)}$, $f_{v(\mu)} = f_{v(g_1)}$, and $l_{v(\mu)} = l_{v(g_q)}$, where f_1 and f_p (g_1 and g_q) are the first and the last virtual edge in the linear list of the virtual edges containing an edge of H and incident to $u(\mu)$ (respectively to $v(\mu)$).

If μ is a P-node, an embedding of skel(μ) is a counterclockwise order of its virtual edges around $u(\mu)$. We describe how to verify if an embedding of skel(μ) exists edge-compatible with $\mathcal{H}$. Consider the virtual edges containing edges of H incident to $u(\mu)$. We show how to construct a list L_u of such edges corresponding to the ordering they have in any embedding of skel(μ) edge-compatible with $\mathcal{H}$. Insert any of such edges, say e_i, into L_u. Repeatedly consider the last element e_j of L_u and insert as new last element of L_u the edge e_{j+1} such that $l(u(e_j))$ immediately precedes $f(u(e_{j+1}))$ in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$. If $e_{j+1} = e_i$, then L_u is the desired circular list. If e_{j+1} does not exist, then the edge following $l(u(e_j))$ belongs to the virtual edge corresponding to the parent of μ. Then, consider the first edge e_i. Repeatedly consider the first element e_j of L_u and insert as new first element of L_u edge e_{j-1} such that $f(u(e_j))$ immediately follows $l(u(e_{j-1}))$ in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$. If e_{j-1} does not exist, then check whether all the virtual edges containing edges of H incident to $u(\mu)$ have been processed and in this case insert the virtual edge corresponding to the parent of μ as first element of L_u. Analogously, construct a list L_v. Let L_{uv} be the sublist obtained by restricting L_u to those edges that appear in L_v. Let L_{vu} be the corresponding sublist of L_v. Check whether L_{uv} and L_{vu} are the reverse of each other. If this is the case, a list L of the virtual edges of skel(μ) containing edges of H incident to $u(\mu)$ or to $v(\mu)$ can be easily constructed compatible with both L_u and L_v. Finally, arbitrarily insert into L the virtual edges of skel(μ) not in L_u and not in L_v, thus obtaining an embedding of skel(μ) edge-compatible with $\mathcal{H}$. Denote by f_1 and f_p (by g_1 and g_q) the virtual edges containing edges of H incident to $u(\mu)$ (respectively to $v(\mu)$) following and preceding the virtual edge representing the parent of μ in L. Set $f_{u(\mu)} = f_{u(f_1)}$, $l_{u(\mu)} = l_{u(f_p)}$, $f_{v(\mu)} = f_{v(g_1)}$, and $l_{v(\mu)} = l_{v(g_q)}$.

Theorem 6.2. *Let $(G, H, \mathcal{H})$ be an n-vertex instance of* Pep *such that both G and H are biconnected. Algorithm BC solves* Pep *for $(G, H, \mathcal{H})$ in $O(n)$ time.*

Proof. We show that Algorithm BC processes each node μ of $\mathcal{T}$ in $O(k_\mu)$ time, where k_μ is the number of children of μ in $\mathcal{T}$.

First, observe that the computation of $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ is trivially done in $O(1)$ time once the embedding of skel(μ) has been decided.

If μ is a Q-node or an S-node, Algorithm BC does not perform any check or embedding choice.

If μ is an R-node, Algorithm BC computes the two planar embeddings of skel(μ) in $O(k_\mu)$ time. For each of these embeddings, Algorithm BC processes each vertex x of skel(μ) separately, considering the list of the virtual edges incident to x (which is trivially constructed in $O(t)$ time, where t is the number of such edges), and restricting the list to those virtual edges containing an edge of H incident to x (for each virtual edge, it suffices to check whether the first edge incident to x is associated with an edge of H, which is done in $O(1)$ time). Checking whether $l_{x(e_i)}$ precedes $f_{x(e_{i+1})}$ in the list of the edges incident to x in $\mathcal{H}$ is done in $O(1)$ time. Hence, the total time spent for each node x is $O(t)$. Summing up over all the nodes of skel(μ) results in a total $O(k_\mu)$ time, as every edge is incident to two nodes and the total number of edges in skel(μ) is $O(k_\mu)$.

If μ is a P-node, extracting the virtual edges of skel(μ) containing edges of H incident to $u(\mu)$ or to $v(\mu)$ can be done in $O(k_\mu)$ time, as in the R-node case. For each of such edges, equipping $f_{u(e)}$, $l_{u(e)}$, $f_{v(e)}$, and $l_{v(e)}$ with a link to e is done in constant time. Determining an ordering of the virtual edges containing edges of H incident to $u(\mu)$ can be done in $O(k_\mu)$ time, as the operations performed for each virtual edge e_i are accessing the first and the last edge of e_i, accessing the edge following the last edge of e_i (preceding the first edge of e_i) in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$, and accessing a virtual edge linked from a first or last edge; each of these operations is trivially done in $O(1)$ time. Marking the virtual edges in L_u and in L_v is done in $O(k_\mu)$ time, as L_u and L_v have $O(k_\mu)$ elements. Then, obtaining L_{uv} and L_{vu}, and checking whether they are the reverse of each other is done in $O(k_\mu)$ time. Finally, extending L_{uv} to L is also easily done in $O(k_\mu)$ time; namely, if L_{uv} is empty, then let L be the concatenation of L_u and L_v (where such lists are made linear by cutting them at any point). Otherwise, start from an edge e_i of L_{uv}; e_i is also in L_u and in L_v; insert e_i into L; insert into L all the edges of L_u following e_i till the next edge e_{i+1} of L_{uv} has been found; insert into L all the edges of L_v preceding e_i till the next edge e_{i+1} of L_{uv} has been found; insert e_{i+1} into L, and repeat the procedure. Each element of L_{uv}, L_u, and L_v is visited once, hence such a step is performed in $O(k_\mu)$ time.

As $\sum_{\mu\in\mathcal{T}} k_\mu = O(n)$, the total running time of the algorithm is $O(n)$. □

Note that, although Algorithm BC relies only on the assumptions that G is biconnected and H is connected, we will only use it in the more special case where H is also biconnected.

6.4.2. G Biconnected, All Vertices and Edges of G Lie in the Same Face of $\mathcal{H}$

The PEGs considered in this section are denoted by $(G(f), H(f), \mathcal{H}(f))$. Such instances are assumed to satisfy the following properties: (i) $G(f)$ is biconnected, (ii) $G(f)$ and $H(f)$ have the same vertex set, (iii) all the vertices and edges of $H(f)$ are incident to the same face f of $\mathcal{H}(f)$, and (iv) no edge of $G(f) \setminus H(f)$ connects two vertices of the same block of $H(f)$. Algorithm BF, that deals with such a setting, is used as a subroutine by Algorithm BA, to be shown later, dealing with the instances of PEP in which G is biconnected and H arbitrary. First, we show that the structure of the cycles in $H(f)$ is very special.

Property 6.3. Every simple path of $H(f)$ belongs to at most one simple cycle of $H(f)$.

Proof. Suppose that there exists a path (that can possibly be a single edge) of $H(f)$ belonging to at least two simple cycles of $H(f)$. Then, such cycles define at least three

regions of the plane. Not all the edges of the two cycles can be incident to the same region, contradicting the fact that all the edges of $H(f)$ are incident to the same region of the plane in $\mathcal{H}(f)$. □

Since all vertices and edges are incident to f, the only relevant cycles for which cycle–vertex constraints have to be checked, are the facial cycles of f. We exploit this particular structure of the input to simplify the test of cycle-compatibility with $\mathcal{H}(f)$ for the skeleton of a node μ of $\mathcal{T}(f)$.

Lemma 6.7. *Consider any node μ of $\mathcal{T}(f)$. Then, an embedding of* skel(μ) *is cycle-compatible with $\mathcal{H}(f)$ if and only if, for every facial cycle $\vec{C}$ of $\mathcal{H}(f)$ whose edges project to a cycle $\vec{C}'$ of* skel(μ)*, no vertex and no edge of* skel(μ) *is to the right of $\vec{C}'$, where $\vec{C}'$ is oriented according to the orientation of $\vec{C}$.*

Proof. By assumption (iii) of the input, all the vertices and edges of $H(f)$ are incident to the same face f of $\mathcal{H}(f)$. By construction, every facial cycle $\vec{C}$ of $H(f)$ is oriented in such a way that f and hence all the vertices of $H(f)$ are to the left of $\vec{C}$. Then, by Lemma 6.3, if the edges of $\vec{C}$ determine a cycle $\vec{C}'$ of virtual edges of skel(μ), all the vertices of skel(μ) that are not in $\vec{C}$ and all the virtual edges of skel(μ) that are not in $\vec{C}'$ and that contain vertices of $G(f)$ have to be to the left of $\vec{C}'$. Finally, all the virtual edges that are not in $\vec{C}'$ and that do not contain any vertex of $G(f)$ (that is, virtual edges corresponding to Q-nodes) have one end-vertex that is not in $\vec{C}$, by assumption (iv) of the input. Such an end-vertex forces the edge to be to the left of $\vec{C}'$. □

In order to find compatible embeddings for the skeletons, we again need to find edge-compatible embeddings, which can be done as in Algorithm BC. Further, to check cycle-compatibility, we need to quickly find the projections of facial cycles of f in the skeletons of the nodes of the SPQR-tree. Since every edge (and every path) can be contained in at most one such cycle, the presence of such a path can be encoded in a single flag. Like Algorithm BC, Algorithm BF starts with the construction of the SPQR-tree $\mathcal{T}(f)$ of $G(f)$, roots it at an arbitrary internal node, and visits $\mathcal{T}(f)$ in bottom-up order in such a way that after a node μ of $\mathcal{T}(f)$ has been visited, an embedding of skel(μ) that is compatible with $\mathcal{H}(f)$ is selected, if it exists. In order to ensure edge compatibility, Algorithm BF maintains edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ for each node μ of $\mathcal{T}(f)$ (and of a virtual edge e) as in Algorithm BC.

Additionally, in order to keep track of paths in H that are part of a facial cycle of f, the algorithm maintains a flag $p(\mu)$ for each node μ of $\mathcal{T}(f)$ such that $p(\mu)$ is set to TRUE if and only if there exists a *traversing path* P, that is, a path between $u(\mu)$ and $v(\mu)$ that is composed of edges of $H(f)$, that belongs to pert(μ), and that is part of a facial cycle $\vec{C}$ of f not entirely contained in pert(μ); flag $p(\mu)$ is set to FALSE, otherwise. Additionally, to encode the direction of the path the algorithm maintains a flag $uv(\mu)$. If $p(\mu) =$ TRUE, the flag $uv(\mu)$ is set to TRUE if P is oriented from $u(\mu)$ to $v(\mu)$ according to the orientation of $\vec{C}$, and it is set equal to FALSE, otherwise. We also refer to these flags as $p(e)$ and $uv(e)$, where e is the virtual edge corresponding to μ in the skeleton of the parent of μ.

Now, we state lemmas specifically dealing with S-, R-, and P-nodes of $\mathcal{T}(f)$. For S-nodes the flags can easily be computed from the flags of its children.

Lemma 6.8. *Let μ be an S-node of $\mathcal{T}(f)$ with children $\mu_1, \mu_2, \ldots, \mu_k$. Then, $p(\mu_i) =$* TRUE *for some $1 \leq i \leq k$ if and only if $p(\mu_j) =$* TRUE *for all $1 \leq j \leq k$.*

Proof. If $p(\mu_j) = \text{TRUE}$ for all $1 \leq j \leq k$, then trivially $p(\mu_i) = \text{TRUE}$. If $p(\mu_i) = \text{TRUE}$ for some $1 \leq i \leq k$, there exists a traversing path of μ_i that is part of a simple cycle $\vec{C}$ of $H(f)$ not entirely contained in $\text{pert}(\mu_i)$; however, as μ is an S-node, $\vec{C}$ does not entirely lie inside $\text{pert}(\mu)$, as otherwise it would entirely lie inside $\text{pert}(\mu_i)$. Then, $\vec{C}$ consists of a traversing path of $\text{pert}(\mu_j)$, for all $1 \leq j \leq k$, and of a traversing path of the virtual edge of $\text{skel}(\mu)$ corresponding to the parent of μ in $\mathcal{T}(f)$, thus proving the lemma. □

Next, we derive a simple criterion for an embedding of an R-node to be cycle-compatible. By Lemma 6.7 an embedding of a skeleton $\text{skel}(\mu)$ is cycle-compatible if for each facial cycle $\vec{C}$ of f that projects to a cycle $\vec{C}'$ in $\text{skel}(\mu)$, the right side is empty. For an R-node μ this condition can be reformulated: each such cycle must have a face on its right side.

Lemma 6.9. *Let μ be an R-node of $\mathcal{T}(f)$. If an edge e of* $\text{skel}(\mu)$ *has a traversing path belonging to a facial cycle $\vec{C}$, let us orient e in the direction determined by the projection of $\vec{C}$ in* $\text{skel}(\mu)$*. An embedding of* $\text{skel}(\mu)$ *is cycle-compatible with $\mathcal{H}(f)$ if and only if, for each face g of the embedding of* $\text{skel}(\mu)$*, either (i) every virtual edge e on the boundary of g is oriented so that g is to the right of e, or (ii) none of the virtual edges on the boundary of g is oriented in a way that g is to the right of it.*

Proof. Suppose that an embedding of $\text{skel}(\mu)$ is cycle-compatible with $\mathcal{H}(f)$. Let g be a face of the embedding. Assume that on the boundary of g there is an edge e containing a traversing path P, such that g is to the right of e. Let $\vec{C}$ be the facial cycle of $\mathcal{H}(f)$ that contains P. By Lemma 6.7, $\vec{C}$ projects to a directed cycle $\vec{C}'$ in $\text{skel}(\mu)$, and no vertex or edge of $\text{skel}(\mu)$ is embedded to the right of $\vec{C}'$. Thus, $\vec{C}'$ corresponds to the boundary of the face g, and hence g satisfies condition (i).

Suppose now that in an embedding of $\text{skel}(\mu)$, every face satisfies condition (i) or condition (ii). We claim that the embedding of $\text{skel}(\mu)$ is cycle-compatible with $\mathcal{H}(f)$. To prove it, we use Lemma 6.7. Let $\vec{C}$ be a facial cycle of $\mathcal{H}(f)$ that projects to a simple cycle $\vec{C}'$ in $\text{skel}(\mu)$. Let e be any edge of $\vec{C}'$ and let g be the face to the right of e in the embedding of $\text{skel}(\mu)$. Necessarily, g satisfies condition (i). Hence, each edge on the boundary of g has a traversing path. The union of these paths forms a cycle in $\mathcal{H}(f)$, and by Property 6.3, this cycle is equal to $\vec{C}$. Thus, the boundary of g coincides with the cycle $\vec{C}'$. In particular, no vertex and no edge of $\text{skel}(\mu)$ is embedded to the right of $\vec{C}'$. By Lemma 6.7, this means that the embedding of $\text{skel}(\mu)$ is cycle-compatible with $\mathcal{H}(f)$. □

It remains to deal with the P-nodes. The special structure of the PEGs $(G(f), H(f), \mathcal{H}(f))$ implies that for each P-node μ there exists at most one facial cycle $\vec{C}$ of f that projects to a cycle in $\text{skel}(\mu)$.

Lemma 6.10. *Let μ be a P-node of $\mathcal{T}(f)$. There exist either zero or two virtual edges of* $\text{skel}(\mu)$ *containing a traversing path.*

Proof. If there exists one virtual edge e_i of $\text{skel}(\mu)$ containing a traversing path that is part of a simple cycle $\vec{C}$ of $H(f)$ not entirely contained in $\text{pert}(e_i)$, another virtual edge of $\text{skel}(\mu)$ containing a traversing path that is part of $\vec{C}$ exists, as otherwise $\vec{C}$ would not be a cycle. Further, if there exist at least three virtual edges of $\text{skel}(\mu)$ containing traversing paths, then each of such paths belongs to three simple cycles, thus contradicting Property 6.3. □

Hence, the skeleton of every P-node contains at most one such cycle, whose right side must by empty by Lemma 6.7. This considerably simplifies the problem of finding a

cycle-compatible embedding of a P-node. We are now ready to exhibit the main steps of Algorithm BF.

Algorithm BF As stated above, Algorithm BF performs a bottom-up traversal on the rooted SPQR-tree $\mathcal{T}(f)$ of $G(f)$, such that for each processed node μ a compatible embedding of skel(μ) is computed, if it exists. It computes the edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ for each node μ of $\mathcal{T}(f)$ (and of a virtual edge e) as in Algorithm BC, in order to find edge-compatible embeddings. Further, it computes the flags $p(\mu)$ and $uv(\mu)$ for each processed node, in order to identify facial cycles of f that project to a cycle in skel(μ). We now give a detailed description how Algorithm BF processes a node μ, assuming that all flags and edges for all children have already been computed.

If μ is a Q- or an S-node, no check is needed. As skel(μ) is a cycle, the only planar embedding of skel(μ) is compatible with $\mathcal{H}(f)$. Edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$, and flags $p(\mu)$ and $uv(\mu)$ can be easily computed.

If μ is an R-node, for each of the two planar embeddings of skel(μ), check if it is edge-compatible with $\mathcal{H}(f)$ and set values for $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ as in Algorithm BC. In order to check if any of the two embeddings is cycle-compatible with $\mathcal{H}(f)$, we check if Lemma 6.7 is satisfied. To perform this test, we need for each virtual edge $e = uv$ of skel(μ) the corresponding flags $p(e)$ and $uv(e)$. This information is already known for all virtual edges, except the virtual edge e_p of skel(μ) representing the parent of μ in $\mathcal{T}(f)$. To compute the flags for e_p, we need to determine whether the virtual edge e_p contains a traversing path P_p and, in case it does, determine its orientation. By definition of traversing path, P_p exists if and only if there exists a traversing path in pert(μ). Restrict skel(μ) to those edges $e_i \neq e_p$ with $p(e_i) = \text{TRUE}$ and denote by skel$'(\mu)$ the obtained graph. Check if the degree of $u(\mu)$ and $v(\mu)$ in skel$'(\mu)$ is even or odd. In the former case, P_p does not exist; set $p(\mu) = \text{FALSE}$ and $p(e_p) = \text{FALSE}$. In the latter case, P_p exists; set $p(\mu) = \text{TRUE}$ and $p(e_p) = \text{TRUE}$; the orientation of P_p is the only one that makes the number of edges e_i incident to $u(\mu)$ with $uv(e_i) = \text{TRUE}$ equal to the number of edges e_i incident to $u(\mu)$ with $uv(e_i) = \text{FALSE}$; this determines $uv(\mu)$ and $uv(e_p)$.

Now, $p(e_i)$ and $uv(e_i)$ are defined for every virtual edge e_i of skel(μ). Consider every face g of skel(μ) and denote by $e_j = (u_j, v_j)$ any edge incident to g. Suppose, without loss of generality, that g is to the right of e_j when traversing such an edge from u_j to v_j. Then, check if $p(e_j) = \text{FALSE}$, or $p(e_j) = \text{TRUE}$ and $uv(e_j) = \text{FALSE}$, for all edges e_j incident to g, and check whether $p(e_j) = \text{TRUE}$ and $uv(e_j) = \text{TRUE}$, for all edges e_j incident to g. If one of the two checks succeeds, the face does not violate Lemma 6.7, otherwise it does.

If μ is a P-node, check if an embedding of skel(μ) exists that is compatible with $\mathcal{H}(f)$ as follows. By Lemma 6.10, there exist either zero or two virtual edges of skel(μ) containing a traversing path. Then, consider the children μ_i of μ such that $p(\mu_i) = \text{TRUE}$. If zero or two such children exist, then the edge of skel(μ) corresponding to the parent ν of μ in $\mathcal{T}$ has no traversing path; if one such a child exists, then the edge of skel(μ) corresponding to ν has a traversing path. Denote by e_i and e_j the edges of skel(μ) containing a traversing path, if such edges exist, where possibly e_j corresponds to ν (in this case, set $p(e_j) = \text{TRUE}$, and set $uv(e_j) = \text{TRUE}$ if $uv(e_i) = \text{FALSE}$ and $uv(e_j) = \text{FALSE}$ otherwise). If there exists no edge e_i of skel(μ) such that $p(e_i) = \text{TRUE}$, then construct an embedding of skel(μ) that is edge-compatible with $\mathcal{H}(f)$, if possible, as in Algorithm BC; as there exists no facial cycle of $\mathcal{H}(f)$ whose edges belong to distinct virtual edges of skel(μ), then an edge-compatible embedding is also cycle-compatible with $\mathcal{H}(f)$. Edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ are computed as in Algorithm BC. Flag $p(\mu) = \text{FALSE}$. If there exist two edges e_i and e_j such that $p(e_i) = \text{TRUE}$, $p(e_j) = \text{TRUE}$, and $p(e_l) = \text{FALSE}$ for every edge $e_l \neq e_i, e_j$, suppose

that $uv(e_i) = \text{TRUE}$ and $uv(e_j) = \text{FALSE}$, the case in which $uv(e_i) = \text{FALSE}$ and $uv(e_j) = \text{TRUE}$ being analogous. Then, by Lemma 6.7, e_j has to immediately precede e_i in the counterclockwise order of the edges incident to $u(\mu)$. Then, construct L_u and L_v as in Algorithm BC; check whether L_u and L_v, restricted to the edges that appear in both lists, are the reverse of each other; further, check whether e_j precedes e_i in L_u and whether e_i precedes e_j in L_v; if the checks are positive construct the list L of all the edges of skel(μ) as in Algorithm BC, except for the fact that the edges of skel(μ) not in L_u and not in L_v are not inserted between e_j and e_i. Edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ are computed as in Algorithm BC. Set $p(\mu) = \text{FALSE}$ if e_j corresponds to a child μ_j of μ and $p(\mu) = \text{TRUE}$ if e_j corresponds to the parent of μ in $\mathcal{T}$; in the latter case, $uv(\mu) = \text{TRUE}$ if $uv(\mu_i) = \text{TRUE}$ and $uv(\mu) = \text{FALSE}$ otherwise. We get the following theorem.

Theorem 6.3. *Let $(G(f), H(f), \mathcal{H}(f))$ be a biconnected* PEG *with n vertices such that $G(f)$ and $H(f)$ have the same vertex set, all the vertices and edges of $H(f)$ are incident to the same face f of $\mathcal{H}(f)$, and no edge of $G(f) \setminus H(f)$ connects two vertices belonging to the same block of $H(f)$. Algorithm BF solves* PEP *for $(G(f), H(f), \mathcal{H}(f))$ in $O(n)$ time.*

Proof. We show that Algorithm BF processes each node μ of $\mathcal{T}(f)$ in $O(k_\mu)$ time, where $\mu_1, \ldots, \mu_{k_\mu}$ are the children of μ in $\mathcal{T}(f)$.

Observe that the computation of $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ and the check of edge-compatibility are done as in Algorithm BC, hence they take $O(k_\mu)$ time. We describe how to check the cycle-compatibility of an embedding of skel(μ) in $O(k_\mu)$ time.

If μ is a Q-node or an S-node, Algorithm BF neither performs any check nor any embedding choice.

If μ is a P-node, then Algorithm BF performs the same checks and embedding choices as Algorithm BC, plus the check that the two edges e_i and e_j with $p(e_i) = \text{TRUE}$ and $p(e_j) = \text{FALSE}$ (notice that one of such edges could be the virtual edge corresponding to the parent of μ) are consecutive (with the right order) in L_u and L_v. This is done in constant time. Flags $p(\mu)$ and $uv(\mu)$ are computed in $O(k_\mu)$ time, by simply checking the flags $p(\mu_i)$ and $uv(\mu_i)$, for $i = 1, \ldots, k$.

Suppose that μ is an R-node. The construction of skel$'(\mu)$ can be easily done in $O(k_\mu)$ time, as such a graph can be obtained from skel(μ) by simply checking flag $p(e_i)$, for each edge e_i in skel(μ). Then, the degree of $u(\mu)$ and $v(\mu)$ in skel$'(\mu)$, and the flags $p(\mu)$, $uv(\mu)$, $p(e_p)$ and $uv(e_p)$ can be computed in total $O(k_\mu)$ time. The test on each face takes time linear in the number of edges incident to the face. Namely, such a test consists of two checks, each of which requires to consider a constant number of flags associated with each edge of the face. As every edge is incident to two faces of skel(μ) and the number of edges in skel(μ) is $O(k_\mu)$, the total time spent for the test on the faces of skel(μ) is $O(k_\mu)$.

As $\sum_{\mu \in \mathcal{T}} k_\mu = O(n)$, the total running time of the algorithm is $O(n)$. □

6.4.3. G Biconnected

In this section we show how to solve PEP for general biconnected PEGs, that is PEGs $(G, H, \mathcal{H})$ where G is biconnected and H is arbitrary. The algorithm employs the algorithms from the previous two sections as subroutines. The general outline is as follows. First, compute a subgraph H^+ of G with the following properties: (i) H^+ is biconnected; (ii) H is a subgraph of H^+; (iii) H^+ contains every non-local H-bridge of G. Second, solve instance $(H^+, H, \mathcal{H})$ obtaining an embedding $\mathcal{H}^+$ of H^+ extending $\mathcal{H}$, if H^+ admits one. We will show that this step can reduced to several applications of Algorithm BF. Finally,

solve instance $(G, H^+, \mathcal{H}^+)$ with Algorithm BC – we will see that H^+ is connected (even biconnected) and hence the algorithm can be applied.

In a first step, we ensure that all non-local H-bridges of G are trivial. Recall that every non-trivial non-local H-bridge K has at most one candidate face f_K of $\mathcal{H}$, where it can be embedded. We obtain the graph H' with embedding $\mathcal{H}'$ as in Definition 6.4 by adding the vertices the $K - V(H)$ to H and embedding them into the face f_K for each non-trivial non-local H-bridge K of G.

Let H^+ be the graph obtained from G by removing the vertices and edges (but not the attachments) of all the local H-bridges of G. Notice that H^+ has the same vertex set as H' and that any embedding of H^+ that extends $\mathcal{H}$ also extends $\mathcal{H}'$ and vice versa.

Each H'-bridge K of H^+ is non-local and therefore there exists a unique face f_K, where it needs to be embedded. Since H-bridges that are embedded in distinct faces of $\mathcal{H}$ do not interact, we can solve the instances stemming from the faces of $\mathcal{H}$ independently, which enables us to use Algorithm BF to find an embedding extension of H^+. This motivates the following definitions, which take a more local view at the Peg $(H^+, H', \mathcal{H}')$. Let f be a face of $\mathcal{H}'$ and let $V(f)$ be the set of vertices of H' that are incident to f. Let $H(f)$ be the subgraph of H' induced by $V(f)$, let $\mathcal{H}(f)$ be $\mathcal{H}'$ restricted to $H(f)$, and let $G(f)$ be the subgraph of H^+ induced by $V(f)$. By construction, in any embedding of H^+ that extends $\mathcal{H}$, the edges of $G(f)$ not belonging to $H(f)$ are embedded inside f.

Our approach is to first find an embedding $\mathcal{H}^+$ of H^+ that extends $\mathcal{H}'$ (that is solve Pep for $(H^+, H', \mathcal{H}')$) and then find an embedding $\mathcal{G}$ for $(G, H^+, \mathcal{H}^+)$ (that is solve Pep for $(G, H^+, \mathcal{H}^+)$). The latter step is actually simple, since H^+ is biconnected and thus connected. Therefore Algorithm BC can be used to solve this subproblem.

Lemma 6.11. *H^+ is biconnected.*

Proof. By construction of H^+, each H^+-bridge of G has all its attachment vertices in the same block of H, and hence in the same block of H^+, as H is a subgraph of H^+. Therefore, the number of blocks of H^+ is not modified by the addition of the H^+-bridges of G. Since such an addition produces G, which is biconnected, H^+ is biconnected. □

Clearly, if $(G, H, \mathcal{H})$ is planar, then an an embedding $\mathcal{G}$ of G extending $\mathcal{H}$ exists, and the restriction of $\mathcal{G}$ to H^+ yields an intermediate embedding $\mathcal{H}^+$ of H^+ extending $\mathcal{H}'$, which can then be extended to an embedding of G extending $\mathcal{H}$. We show that the choice of $\mathcal{H}^+$ does not change the possibility of finding such an embedding extension. In particular, if $\mathcal{H}_1^+$ and $\mathcal{H}_2^+$ are two embeddings of $(H^+, H, \mathcal{H})$ then the Peg $(G, H^+, \mathcal{H}_1^+)$ is planar if and only if $(G, H^+, \mathcal{H}_2^+)$ is planar.

Lemma 6.12. *A biconnected* Peg $(G, H, \mathcal{H})$ *is planar if and only if (a) the* Peg $(H^+, H, \mathcal{H})$ *admits a planar embedding and (b) for every such embedding $\mathcal{H}^+$, $(G, H^+, \mathcal{H}^+)$ is planar.*

Proof. Clearly, if conditions (a) and (b) hold, then G has an embedding extending $\mathcal{H}$.

To prove the converse, assume that G has an embedding $\mathcal{G}$ extending $\mathcal{H}$. Clearly, $\mathcal{G}$ contains a sub-embedding $\mathcal{H}^+$ of H^+ that extends $\mathcal{H}$, so condition (a) holds. It remains to prove that condition (b) holds, too.

First, we introduce some terminology: Let f be any face of $\mathcal{H}$ and let $\mathcal{H}^+$ be any embedding of H^+ that extends $\mathcal{H}$. In $\mathcal{H}^+$, the face f can be partitioned into several faces, which we will call the *subfaces of f*. A set of vertices $S \subseteq V(H)$ is said to be *mutually visible in f with respect to $\mathcal{H}^+$* if $\mathcal{H}^+$ has a subface of f that contains all the vertices of S on its boundary.

The proof that condition (b) holds is based on two claims. The first one shows that for the vertices that belong to the same block of H, mutual visibility is independent of the choice of $\mathcal{H}^+$.

Claim 1. Let $\vec{C}$ be a facial cycle of f and let $S \subseteq V(\vec{C})$ be a set of vertices of $\vec{C}$. If the vertices in S are mutually visible in f with respect to at least one embedding of H^+ that extends $\mathcal{H}$, then they are mutually visible in f with respect to every embedding of H^+ that extends $\mathcal{H}$.

Note that the mutual visibility of S in f only depends on the embedding $\mathcal{H}^+$ restricted to $G(f)$. Let $\mathcal{T}$ be the SPQR-tree of $G(f)$. By Theorem 6.1, the embeddings of $G(f)$ that extend $\mathcal{H}(f)$ are exactly obtained by specifying a compatible embedding for each skeleton of $\mathcal{T}$. Assume that $\mathcal{G}_1$ and $\mathcal{G}_2$ are two embeddings of $G(f)$ that extend $\mathcal{H}$. Assume that the vertices of S are mutually visible in f with respect to $\mathcal{G}_1$. We will show that they are also mutually visible with respect to $\mathcal{G}_2$. In view of Theorem 6.1, we may assume that $\mathcal{G}_2$ was obtained from $\mathcal{G}_1$ by changing the embedding of the skeleton of a single node $\mu \in \mathcal{T}$.

Let us distinguish two cases, depending on whether the cycle $\vec{C}$ is contained in the pertinent graph of a single edge of μ, or whether it projects to a cycle in μ. If $\vec{C}$ is part of the pertinent graph of a single virtual edge $e = \{x, y\} \in \mu$, then let $\mathcal{G}_e$ be the embedded graph obtained as the union of the pertinent graph of e and a single edge connecting x and y, embedded in the outer face of the pertinent graph. We easily see that the vertices S are mutually visible in f if and only if they share the same face of $\mathcal{G}_e$, other than the face that is to the right of $\vec{C}$. Since $\mathcal{G}_e$ does not depend on the embedding of μ, we see that S are mutually visible in $\mathcal{G}_2$.

Assume now that the cycle $\vec{C}$ projects to a cycle $\vec{C}'$ in μ. By Lemma 6.7, in any compatible embedding of μ, all the vertices and edges of μ that do not belong to $\vec{C}'$ are embedded to the left of $\vec{C}'$. In particular, if μ is an R-node, it only has a single compatible embedding. Thus, μ must be a P-node. Let e and e' be the two virtual edges of μ that form $\vec{C}'$. In each compatible embedding of μ, these two edges must be embedded next to each other, and in the same order. It easily follows that any two compatible embeddings of μ yield embeddings of $G(f)$ in which the vertices from S have the same mutual visibility. This completes the proof of the claim.

Let us proceed with the proof that condition (b) holds. We need more terminology: Let K and K' be a pair of local H-bridges of G whose attachments all appear on a facial cycle $\vec{C}$ of a face f in $\mathcal{H}$. We say that K and K' have a *three-vertex conflict on $\vec{C}$* if they share at least three attachments, and that they have a *four-vertex conflict on $\vec{C}$* if there are four vertices x, x', y, y' that appear on $\vec{C}$ in this cyclic order, and x, y are attachments of K, while x', y' are attachments of K'.

Claim 2. Assume that a face f_K of $\mathcal{H}$ has been assigned to every local H-bridge K of G so that all the attachments of K are on the boundary of f_K. Let $\mathcal{H}^+$ be an embedding of H^+ extending $\mathcal{H}$. There is an embedding $\mathcal{G}$ of G extending $\mathcal{H}^+$, with the additional property that each local H-bridge K is embedded inside a subface of f_K, if and only if:

1. *For any local H-bridge K, all the attachments of K are mutually visible in f_K with respect to $\mathcal{H}^+$.*
2. *If K and L are distinct local H-bridges assigned to the same face $f_K = f_L$, such that the attachments of K and L appear on a common facial cycle $\vec{C}$ of $\mathcal{H}^+$, then K and L have no conflict on $\vec{C}$.*

Clearly, the two conditions are necessary. In order to prove that they are also sufficient, assume that both the conditions hold. Construct an embedding of $\mathcal{G}$ with the desired

properties as follows. Let f be any face of $\mathcal{H}$ and let f' be a face of $\mathcal{H}^+$ that is a subface of f. Let $K_1, \dots K_s$ be all the local H-bridges that were assigned to f and such that all their attachments appear on the boundary of f'. Observe that the first condition of the claim guarantees that every H-bridge K_i can be assigned to a face f' such that all the attachments of K_i are mutually visible in f'. We show that all the bridges $K_1, \dots K_s$ can be embedded inside f'.

First, observe that the boundary of f' is a simple cycle C', because H^+ is biconnected. Observe also that no two bridges K_i and K_j have a conflict on C', by the second condition of the claim. To show that all the bridges $K_1, \dots, K_s$ can be embedded inside C', proceed by induction on s. If $s = 1$ the statement is clear. Assume that $s \geq 2$ and that the bridge K_1 has been successfully embedded into f'. The embedding of K_1 partitions f' into several subfaces $f'_1, \dots, f'_t$. Such subfaces are again bounded by simple cycles, otherwise G would not be biconnected. We claim that, for every bridge K_i, with $i \geq 2$, there is a subface f'_j containing all the attachments of K_i. Consider any bridge K_i. Assume first that K_i has an attachment x that is not an attachment of K_1. Then, x belongs to a unique subface f'_j. Hence, if K_i has another attachment not belonging to f'_j, there is a four-vertex conflict of K_1 and K_i on $\vec{C}'$, contradicting the second condition of the claim. Assume next that each attachment of K_i is also an attachment of K_1. Then, K_i has exactly two attachments and, if such attachments do not share a face f'_j, a four-vertex conflict of K_1 and K_i on $\vec{C}'$ is created, again contradicting the second condition of the claim.

We can thus assign to each K_i a subface f'_j that contains all its attachments. By induction, all the K_i's can be embedded into their assigned faces, thus proving the second claim.

The proof that condition (b) holds easily follows from the two claims. Namely, assume that G has an embedding $\mathcal{G}$ extending $\mathcal{H}$. Let $\mathcal{H}^+$ be $\mathcal{G}$ restricted to H^+. For every local H-bridge K of G, let f_K be the face of $\mathcal{H}$ inside which K is embedded in $\mathcal{G}$. Clearly, $\mathcal{H}^+$ satisfies the two conditions of the second claim, since it can be extended into $\mathcal{G}$. Then, every embedding of H^+ that extends $\mathcal{H}$ satisfies the two conditions of the second claim: For the first condition, this is a consequence of the first claim and for the second condition this is obvious. We conclude that every embedding of H^+ that extends $\mathcal{H}$ can be extended into an embedding of G, thus proving condition (b) and hence the lemma. □

As stated above, for each H' bridge K of $(H^+, H', \mathcal{H}')$ is non-local and we therefore know into which face f_K it needs to be embedded. Since H'-bridges that are embedded in different faces do not interact, we can solve the subinstance $(G(f), H(f), \mathcal{H}(f))$ arising from each face, separately. Clearly, if one of the instances fails, then G does not have an embedding extension. If all instances admit an embedding extension, gluing them together yields an embedding $\mathcal{H}^+$ of H^+ extending $embH'$. The previous lemma then implies that $(G, H, \mathcal{H})$ is planar if and only if $(G, H^+, \mathcal{H}^+)$ is planar. We are now ready to describe Algorithm BA (for G **B**iconnected and H **A**rbitrary).

Algorithm BA Starting from an instance $(G, H, \mathcal{H})$ of Pep, graphs $G(f)$ and $H(f)$, and embedding $\mathcal{H}(f)$, for every face f of $\mathcal{H}$, are computed as follows. For each H-bridge K of G, determine whether it is local to a block of H or not. In the former case, K is not associated to any face f of $\mathcal{H}$. In the latter case, we compute the unique face f of $\mathcal{H}$ in which K has to be embedded in any solution of instance $(G, H, \mathcal{H})$ of Pep and we associate K with f. These computations involve checks on the $\mathcal{CF}$-tree of $\mathcal{H}$, on the $\mathcal{BF}$-tree of $\mathcal{H}$, on the $\mathcal{VF}$-graph of $\mathcal{H}$, and on the enriched block-cutvertex tree of each connected component of H. However, all these computations can be performed in time

linear in the size of K, as shown in the following.

Lemma 6.13. *Let $(G, H, \mathcal{H})$ be any instance of* Pep*. Let K be an H-bridge of G. There is an algorithm that checks whether K is local to any block of H in time linear in the size of K. Further, if K is non-local, the algorithm computes the only face of $\mathcal{H}$ incident to all the attachment vertices of K, if such a face exists, in time linear in the size of K.*

Proof. Compute the component-face tree $\mathcal{CF}$ of $\mathcal{H}$, rooted at any node, the vertex-face incidence graph $\mathcal{VF}$ of $\mathcal{H}$, the block-face tree $\mathcal{BF}$ of $\mathcal{H}$, rooted at any node, and, for each connected component C_i of H, the enriched block-cutvertex tree $\mathcal{B}_i^+$ of C_i, rooted at any node. These computations can be performed in linear time (as shown in the Data Structures section, Section 6.2.3).

Consider the attachment vertices $a_1, a_2, \ldots, a_h$ of K. If $h = 1$, then K is local. Otherwise, $h \geq 2$. In order to decide whether K is local for some block of H, we perform the following check. Consider the attachment vertices a_1 and a_2. If a_1 and a_2 belong to distinct connected components, then K is not local to any block. Otherwise, they belong to the same connected component C_i. Check whether a_1 and a_2 have distance 2 in $\mathcal{B}_i^+$, that is, whether they belong to the same block B. This can be done in constant time [KK03]. If the check fails, then K is not local to any block. Otherwise, B contains both a_1 and a_2. In the latter case, check whether B is also adjacent in $\mathcal{B}_i^+$ to all the other attachment vertices $a_3, \ldots, a_h$ of K. Again, each such a check is performed in constant time [KK03]. If the test succeeds, then K is local to block B. Otherwise, there exists a vertex a_j, with $3 \leq j \leq h$, that is not incident to B, and K is not local to any block.

If K is non-local, we compute the unique face f of $\mathcal{H}$ to which all the attachment vertices of K are incident. First, we choose two attachment vertices a_p and a_q, with $1 \leq p, q \leq h$, that do not belong to the same block. If a_1 and a_2 do not belong to the same block, then we take $a_p = a_1$ and $a_q = a_2$. If the check failed on an attachment vertex a_j in $a_3, \ldots, a_h$, then either a_1 and a_j, or a_2 and a_j do not belong to the same block. In the former case set $a_p = a_1$ and $a_q = a_j$, in the latter one $a_p = a_2$ and $a_q = a_j$. Since the vertex-face incidence graph $\mathcal{VF}$ is planar, we may use the approach of [KK03] to determine in constant time whether a_p and a_q are connected by a path of length two in $\mathcal{VF}$, and find the middle vertex of such a path. This middle vertex corresponds to the unique common face f of a_p and a_q. Check whether all the attachments of K are adjacent to f in $\mathcal{VF}$. If the test fails, then no face of $\mathcal{H}$ contains all the attachments of K. Otherwise, f is the only face of $\mathcal{H}$ whose boundary contains all the attachments of K. □

For each face f of $\mathcal{H}$, consider every H-bridge K associated with f. Add the vertices and the edges of K to $G(f)$, and add the vertices of K to $\mathcal{H}(f)$ inside f. Let $H^+ = \bigcup_{f \in H} G(f)$. For each face f of $\mathcal{H}$ call Algorithm BF with input $(G(f), H(f), \mathcal{H}(f))$. If Algorithm BF succeeds for every instance $(G(f), H(f), \mathcal{H}(f))$ (thus providing an embedding $\mathcal{H}^+(f)$ of $G(f)$ whose restriction to $H(f)$ is $\mathcal{H}(f)$), merge the embeddings $\mathcal{H}^+(f)$ of $G(f)$ into a planar embedding $\mathcal{H}^+$ of H^+. Finally, call Algorithm BC with $(G, H^+, \mathcal{H}^+)$.

Theorem 6.4. *Let $(G, H, \mathcal{H})$ be an n-vertex instance of* Pep *such that G is biconnected. Algorithm BA solves* Pep *for $(G, H, \mathcal{H})$ in $O(n)$ time.*

Proof. The correctness of the algorithm descends from Lemma 6.12.

By Lemma 6.13, determining whether an H-bridge K is local or not can be done in time linear in the size of K. Further, if K is non-local, the only face of $\mathcal{H}$ incident to all the attachment vertices of K can be computed, if it exists, in time linear in the size of K.

Then, the construction of graphs $G(f)$, $H(f)$, H^+ and of embeddings $\mathcal{H}(f)$ takes $O(n)$ time, as it only requires to perform the union of graphs that have total $O(n)$ edges.

By Theorem 6.3, Algorithm BF runs in time linear in the number of edges of $G(f)$, hence all the executions of Algorithm BF take a total $O(n)$ time. By Theorem 6.2, Algorithm BC runs in $O(n)$ time, hence the total running time of Algorithm BA is $O(n)$. □

This concludes the case of biconnected PEGs.

6.4.4. G Connected or Disconnected

In this section we give an algorithm that decides the planarity of general PEGs. First, we deal with instances $(G, H, \mathcal{H})$ of PEP in which G is connected, every non-trivial H-bridge of G is local, and H is arbitrary. We show that the three conditions of Lemma 6.4 can be checked in linear time. The first condition can be checked in linear time by Lemma 6.13. The second and the third conditions can be checked in linear time by the following two lemmas.

Lemma 6.14. *Let $(G, H, \mathcal{H})$ be a connected* PEG*. Let $G_1, \ldots, G_t$ be the blocks of G, and let H_i be the subgraph of H induced by the vertices of G_i. There is a linear-time algorithm that checks whether any two distinct graphs among $H_1, \ldots, H_t$ alternate around a vertex of $\mathcal{H}$.*

Proof. Let us describe the algorithm that performs the required checks. We assume that every edge e of H has an associated label indicating the block of G that contains e. We also associate to each block two integer counters which will be used in the algorithm.

We now describe a procedure TEST(x) which, for a given vertex $x \in V(H)$, checks whether any two graphs H_i, H_j alternate around x. Let us use the term *x-edge* to refer to any edge of H incident to x, and let *x-block* refer to any block of G that contains at least one x-edge.

The procedure TEST(x) proceeds as follows: first, for every x-block G_i, it determines the number of x-edges in G_i and stores this in a counter associated with G_i. This is done by simply looking at every edge incident to x and incrementing the counter of the corresponding block. Next, TEST(x) visits all the x-edges in the order determined by the rotation scheme $\sigma_{\mathcal{H}}(x)$, starting at an arbitrary x-edge. For each x-block it maintains in a counter the number of its x-edges that have been visited so far. An x-block is *active* if some but not all of its x-edges have already been visited.

The procedure TEST(x) also maintains a stack containing the active x-blocks. At the beginning of the procedure the counters of visited edges of each x-block are set to zero and the stack is empty.

For every edge e that TEST(x) visits, it performs the following steps:

1. Let G_i denote the block containing e. Increment the counter of visited x-edges of G_i.

2. If no other edge of G_i has been visited so far, push G_i on the stack.

3. If some x-edge of G_i has been visited before e, we know that G_i is currently somewhere on the stack. Check whether G_i is on the top of the stack. If the top of the stack contains an x-block G_j different from G_i, output that H_i and H_j alternate around x and stop.

4. Check whether e is the last x-edge of G_i to be visited (comparing its counter of visited x-edges to the counter of total x-edges), and if it is, pop G_i from the stack. (Note that if G_i has only one x-edge, it is pushed and popped during the visit of this edge.)

If TEST(x) visits all the x-edges without rejecting, it outputs that there is no alternation around x.

The procedure TEST(x) takes time proportional to the number of x-edges. Thus, we can call TEST(x) for all the vertices $x \in V(H)$ in linear time to test whether there is any alternation in $\mathcal{H}$.

Let us now argue that the procedure TEST(x) is correct. Assume that TEST(x) outputs an alternation of H_i and H_j. This can only happen when G_j is on the top of the stack while an x-edge $e \in G_i$ is visited, and furthermore, e is not the first edge of G_i to be visited. It follows that the first edge of G_i was visited before the first edge of G_j, and G_j is still active when e is visited. This shows that H_i and H_j indeed alternate around x.

Conversely, assume that there is a pair of graphs H_i and H_j that alternate around x, and the alternation is witnessed by two pairs of x-edges $e, e' \in H_i$ and $f, f' \in H_j$. For contradiction, assume that TEST(x) outputs that there is no alternation. Without loss of generality, assume that at least one x-edge of H_i is visited before any x-edge of H_j, that e is visited before e', and that f is visited before f'. Thus, the four x-edges are visited in the order e, f, e', f'. When the procedure visits e', both G_i and G_j are active, and G_j is on the stack above G_i, since we assumed that the first x-edge of G_i is visited before the first x-edge of G_j. This means that when TEST(x) visited e', G_i was not on the top of the stack and an alternation should have been reported.

This contradiction completes the proof of the lemma. □

The next lemma shows that the third condition of Lemma 6.4 can also be tested in linear time, assuming the first and second conditions of the lemma hold.

Lemma 6.15. *Let $(G, H, \mathcal{H})$ be a connected* PEG*. Let $G_1, \ldots, G_t$ be the blocks of G, and let H_i be the subgraph of H induced by the vertices of G_i. Let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Assume that the following conditions hold.*

1) each non-trivial H-bridge of G is local,

2) each G_i has an embedding that extends $\mathcal{H}_i$, and

3) no two of the graphs $H_1, \ldots, H_t$ alternate around any vertex of H.

There is a linear-time algorithm that decides whether there exists a facial cycle $\vec{C}$ of $\mathcal{H}$ that separates a pair of vertices x and y such that x and y are connected by a path of G that has no vertex in common with $\vec{C}$.

Proof. Let P be a path in G with end-vertices in H and let $\vec{C}$ be a facial cycle of $\mathcal{H}$. If P and $\vec{C}$ are vertex-disjoint and the end-vertices of P are separated by $\vec{C}$, we say that P and $\vec{C}$ form a *PC-obstruction*. A PC-obstruction $(P, \vec{C})$ is called *minimal* if no proper subpath $P' \subset P$ forms a PC-obstruction with $\vec{C}$. Observe that, in a minimal PC-obstruction, all the internal vertices of P belong to $V(G) \setminus V(H)$.

We want to show that the existence of a PC-obstruction can be tested in linear time. Of course, it is sufficient to test the existence of a minimal PC-obstruction. Before we explain how this test is done, we make some more observations concerning the structure of minimal PC-obstructions.

Let $(P, \vec{C})$ be a minimal PC-obstruction, and let x and y be the end-vertices of P. As the internal vertices of P belong to $V(G) \setminus V(H)$, then P is a subset of an H-bridge K, and x and y are among the attachments of K. Let us now distinguish two cases, depending on whether K is local to some block or not.

First, assume that K is local to a block B of H. Then, both B and P are part of the same block G_i of G. Hence, $\vec{C}$ belongs to a different block of G, because if it belonged to G_i, then G_i would contain the whole PC-obstruction $(P, \vec{C})$ and it would be impossible to extend the embedding $\mathcal{H}_i$ to G_i, thus contradicting condition 2 of the lemma. Then, let G_j be the block of G that contains $\vec{C}$. Since x and y belong to a common block B of H, they are connected by a path $Q \subseteq B$. Since x and y are separated by $\vec{C}$, Q shares a vertex z with $\vec{C}$ (otherwise the embedding $\mathcal{H}$ would not be planar). Since Q and $\vec{C}$ belong to distinct blocks, z is their unique common vertex. Hence, in the rotation scheme of z, the two edges that belong to Q alternate with the two edges that belong to $\vec{C}$, because $\vec{C}$ separates x and y. Thus, G_i alternates with G_j around z, contradicting condition 3 of the lemma. Then, K cannot be a local bridge.

Second, assume that K is non-local. By condition 1 of the lemma, K consists of a single edge of $E(G) \setminus E(H)$.

We conclude that any minimal PC-obstruction $(P, \vec{C})$ has the property that P is a single edge that forms a non-local H-bridge of G.

Observe that two vertices x and y belonging to distinct blocks of H are separated by a facial cycle of $\mathcal{H}$ if and only if there is no face of $\mathcal{H}$ to which both x and y are incident.

We are now ready to describe the algorithm that determines the existence of a minimal PC-obstruction. The algorithm tests all the edges of $E(G) \setminus E(H)$ one by one. For any such an edge e, it determines in constant time whether it is an H-bridge, i.e., whether its endpoints x and y belong to H. If it is an H-bridge, it checks whether it is non-local in constant time, by using Lemma 6.13. For a non-local bridge, the algorithm then checks in constant time whether there is a face f of H into which this bridge can be embedded, again using Lemma 6.13. Such a face f, if it exists, is uniquely determined. Finally the algorithm checks whether both x and y are incident to f, using the vertex-face incidence graph $\mathcal{VF}$.

Overall, for any edge e, the algorithm determines in constant time whether this edge is a non-local bridge that is part of a minimal PC-obstruction. Thus, in linear time, we determine whether G has any PC-obstruction. □

Combining Lemmas 6.4, 6.13, 6.14 and 6.15 with Theorem 6.4, we obtain the following result.

Theorem 6.5. Pep *can be solved in linear time when restricted to instances* $(G, H, \mathcal{H})$ *where* G *is connected.*

Proof. By Lemma 6.13, an instance of Pep where G is connected can be reduced in linear time to an equivalent instance that has the additional property that all the non-trivial H-bridges are local. Namely, whether an H-bridge K is non-local and, in such a case, which is the face of $\mathcal{H}$ in which K has to be embedded can be computed in time linear in the size of K, by Lemma 6.13. We may thus assume that $(G, H, \mathcal{H})$ is an instance of Pep where G is simply connected and all non-trivial H-bridges in G are local to some block.

To solve Pep for $(G, H, \mathcal{H})$, we present an algorithm based on the characterization of Lemma 6.4. First, we generate all the subinstances $(G_i, H_i, \mathcal{H}_i)$ for $i = 1, \ldots, t$, induced by the blocks of G. It is not difficult to see that the subinstances can be generated in linear

time. We then solve these subinstances using Algorithm BA, which takes linear time, by Theorem 6.4, since the total size of the subinstances is linear. If any of the subinstances does not have an embedding extension, we reject $(G, H, \mathcal{H})$, otherwise we continue.

In the next step, we check whether there is a pair of graphs H_i, H_j that have an alternation around a vertex of $\mathcal{H}$. If there is an alternation, we reject the instance, otherwise we continue. This step can be implemented in linear time, due to Lemma 6.14.

Finally, we check the existence of PC-obstructions, which by Lemma 6.15 can be done in linear time. We accept the instance if and only if we find no PC-obstruction. The correctness of this algorithm follows from Lemma 6.4. □

Next, we deal with the instances $(G, H, \mathcal{H})$ of Pep in which G is disconnected and H arbitrary. We use Lemma 6.5 directly, and show that the two conditions of the lemma can be checked in linear time. The first condition of Lemma 6.5 can be checked in linear time by Theorem 6.5. To check the second condition, the $\mathcal{CF}$ tree of $\mathcal{H}$ is considered and rooted at any node representing a face; then, the embedding $\mathcal{H}_i$ is considered as $\mathcal{H}$ restricted to the subgraph H_i of H induced by the vertices of G_i; then, for every i, each node of $\mathcal{CF}$ that represents a face of $\mathcal{H}$ incident to a component of H_i and whose parent represents a component of H not in H_i is considered; if there is more than one such node for some i, then $(G, H, \mathcal{H})$ admits no solution, otherwise it does. The correctness of this argument and an efficient implementation of it are in the proof of the following theorem.

Theorem 6.6. Pep *can be solved in linear time.*

Proof. Let $(G, H, \mathcal{H})$ be an instance of Pep. Let $G_1, \ldots, G_t$ be the connected components of G, let H_i be the subgraph of H induced by the vertices of G_i, and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i.

By Lemma 6.5, $(G, H, \mathcal{H})$ has an embedding extension if and only if each instance $(G_i, H_i, \mathcal{H}_i)$ has an embedding extension and, for $i \neq j$, no facial cycle of $\mathcal{H}_i$ separates a pair of vertices of H_j. By Theorem 6.5, we can test in linear time whether all the instances $(G_i, H_i, \mathcal{H}_i)$ have an embedding extension.

It remains to test the existence of a facial cycle of $\mathcal{H}_i$ that separates vertices of H_j. For this test, we use the component-face tree $\mathcal{CF}$ of $\mathcal{H}$. Assume that $\mathcal{CF}$ is rooted at any node representing a face of $\mathcal{H}$; call this face the *root face of* $\mathcal{H}$. A face f is an *outer face of* $\mathcal{H}_j$ if at least one child of f in $\mathcal{CF}$ is a component of H_j, but the parent of f does not belong to H_j (which includes the possibility that f is the root face).

We claim that a pair of vertices of H_j is separated by a facial cycle belonging to another component of H if and only if there are at least two distinct outer faces of $\mathcal{H}_j$ in $\mathcal{CF}$. To see this, assume first that $\mathcal{H}_j$ has two distinct outer faces f_1 and f_2, and let C_1 (or C_2) be a component of H_j which is a child of f_1 (or f_2, respectively). Any path from C_1 to C_2 in $\mathcal{CF}$ visits the parent of f_1 or the parent of f_2. These parents correspond to components of H not belonging to H_j, and at least one facial cycle determined by these components separates C_1 from C_2.

Conversely, if C_1 and C_2 are components of H_j separated by a facial cycle belonging to a component C_3 of H_i $(i \neq j)$, then the path in $\mathcal{CF}$ that connects C_1 to C_2 visits C_3, and in such a case it is easy to see that $\mathcal{H}_j$ has at least two outer faces.

We now describe the algorithm that tests the second condition of Lemma 6.5. We assume that each connected component of H has associated its corresponding subgraph H_i in $\mathcal{CF}$. We then process the components of H one by one and, for each component C, we check whether its parent node is an outer face of the embedding $\mathcal{H}_i$ of the subgraph

H_i containing C. We accept $(G, H, \mathcal{H})$ if and only if each $\mathcal{H}_i$ has one outer face. This algorithm clearly runs in linear time. □

The algorithms for PEP we presented in this section, the ones for handling 1-connected and disconnected graphs, are non-constructive. For simplicity, we preferred to first present a shorter, non-constructive version. We now briefly sketch how the algorithms can be extended to constructive linear-time algorithms.

Sketch of Constructive Algorithms. For the reduction from disconnected to connected PEGs this is rather simple. Let $(G, H, \mathcal{H})$ be a PEG and let $G_1, \ldots, G_t$ be the connected components of G. Assume we already have an embedding $\mathcal{G}_i$ for each instance $(G_i, H_i, \mathcal{H}_i)$, where H_i is the subgraph of H contained in G_i and $\mathcal{H}_i$ is the restriction of $\mathcal{H}$ to H_i. Once the checking algorithm has been run, we for each connected component of G_i its unique outer face f_i in $\mathcal{H}$. Let f be any face of $\mathcal{H}$ that is an outer face of connected components $G_1, \ldots, G_k$ and possibly has parent G_{k+1} in the rooted component-face tree $\mathcal{CF}$, if f is not the root of $\mathcal{CF}$. We find a subface g_i of f in any of the embedding $\mathcal{G}_1, \ldots, \mathcal{G}_k$, and possibly G_{k+1}, which is clearly possible in time linear in the size of G_i. We obtain the embedding by merging the boundaries of the faces g_i, $i = 1, \ldots, k$, and possibly g_{k+1}. Since every connected components occurs at most once with its outer face and at most once as the parent of an outer face, the total running time is linear.

For the case of 1-connected PEGs, observe that the procedure TEST(x) described in the proof of Lemma 6.14 not just checks whether around each cutvertex x the blocks containing an H-edge incident to x have a parenthetical structure, but the algorithm can actually be employed to find an ordering of these incident blocks $B_1, \ldots, B_t$ of G such that when removing these blocks one by one, the H-edges of B_i form an interval at the time of its removal. Further the check whether a trivial H-bridge is part of a PC-obstruction actually reveals a unique face of $\mathcal{H}$ into which the block containing the H-bridge has to be embedded. We use an arbitrary such H-bridge to determine the correct face for blocks that do not contain an H-edge incident to the cutvertex x. This either gives a correct embedding or one of the conditions of Lemma 6.14 or Lemma 6.15 is violated, in which case an embedding does not exist. In the following, we assume that we have a feasible instance.

Our procedure for handling 1-connected graphs is as follows. We first compute the block-cutvertex tree of G and use Lemma 6.13 to make all non-local H-bridges trivial. Let $G_1, \ldots, \mathcal{G}_t$ be the blocks of G and let $H_1, \ldots, H_t$ be the subgraphs of H induced by $G_1, \ldots, G_t$, together with the embeddings $\mathcal{H}_1, \ldots, \mathcal{H}_t$ induced by $\mathcal{H}$. First, we compute in linear time an embedding extension for each instance $(G_i, H_i, \mathcal{H}_i)$. Next, we merge these embeddings into a single embedding $\mathcal{G}$ extending $\mathcal{H}$, if possible. To this end, we need to merge the rotation schemes of the embeddings at the cutvertices of G. In a first step, we iteratively remove all leaf blocks in the block-cutvertex tree that do not contain a vertex of H distinct from their (unique) cutvertex. Clearly, these blocks can later easily be embedded as they are not subject to any constraints. The remaining instance is still connected by the assumption that all non-local H-bridges are trivial, all cutvertices of the remaining instance belong to H. Therefore, every block contains at least two vertices of H (every non-leaf has at least two cutvertices, and a leaf with only one H-vertex would be removed).

Let x be a cutvertex of the remaining graph with incident blocks $B_1, \ldots, B_t$. For each of them we already have embedding extensions $\mathcal{B}_\rangle$. Assume that the ordering of the blocks is such that $B_1, \ldots, B_k$ contain an H-edge incident to x, while $B_{k+1}, \ldots, B_t$ do not.

We embed the blocks $B_1, \ldots, B_k$ by using a modified version of the procedure TEST(x), described in the proof of Lemma 6.14. For each i in $k+1, \ldots, t$, let $e = xy$ be any edge incident to x in B_i. Since B_i contains an H-vertex distinct from x, the vertex y belongs to H as well, otherwise e would be part of a non-trivial non-local H-bridge. Hence x and y belong to distinct connected components of H. We use the component face tree $\mathcal{CF}$ to find in $O(1)$ time the unique face $_i$ of $\mathcal{H}$ that is shared by x and y. We associate B_i with f_i. Although the choice of e is arbitrary, either all edges of B_i incident to x yield the same face, or at least one of them is part of a PC-obstruction, which we can rule out by first running the checking algorithm.

We now construct the global (cyclic) ordering of all edges incident to x by a single traversal of $\sigma_{\mathcal{H}}(x)$, similar to the procedure TEST(x), except that we alternatingly visit edges and faces as they occur in counterclockwise order around x in $\mathcal{H}$.

When the procedure visits a face f, it appends the edges of all blocks associated with this face in the order as they occur in the embedding of the block. More precisely, let B_i be any block associated with f, let e be any edge of B_i incident to x and let e' be the predecessor of e in the counterclockwise cyclic ordering of x in $\mathcal{B}_\rangle$. Then, the cyclic ordering of x in $\mathcal{B}_\rangle$ forms a sequence $e, \ldots, e'$, which we append to our global ordering of x. We do this for all blocks associated with f in an arbitrary order. When the procedure encounters the first H-edge of a block B_i, it appends this edge to the global ordering of x, and stores the last encountered H-edge of B_i as e_i. Whenever, it encounters another H-edge e' of B_i, it appends all edges between e_i and e', excluding e_i and including e', to the global ordering of x in $\mathcal{B}_\rangle$, and then updates e_i to e'. When the procedure encounters the last H-edge of a block, it also inserts all the remaining edges of $\mathcal{B}_\rangle$ between the last encountered H-edge and the first H-edge of B_i. As the H-edges incident to x occur in the embedding $\mathcal{B}_\rangle$ of each block B_i in the same order as in $\sigma_{\mathcal{H}}(x)$, each edge is inserted exactly once into the cyclic ordering. Considering the output sequence as a cyclic sequence, we have found a cyclic ordering of all incident edges of x. Clearly, the running time of the procedure is proportional to the number of edges incident to x. Clearly, the ordering is such that its restriction to H yields $\sigma_{\mathcal{H}}(x)$, no two blocks alternate, and the ordering of the edges of each incident block B_i are compatible with $\mathcal{B}_\rangle$. Finally, also the blocks that do not have an H-edge incident to x are embedded into the right face, since this face exists and thus is uniquely determined. The previously removed blocks containing at most one H-vertex, can be embedded into arbitrary faces incident to their cutvertices in reverse order of removal. Clearly, the total running time of this procedure is linear. The following theorem summarizes our results.

Theorem 6.7. *Let $(G, H, \mathcal{H})$ be a* PEG*. There is a linear-time algorithm that either finds an embedding extension $\mathcal{G}$ of $\mathcal{H}$ or concludes that such an embedding does not exist.*

6.5. Applications and Extensions

In this section we discuss several extension of the problem PARTIALLYEMBEDDEDPLANARITY. Additionally, we show that PEP has some connections to the problem of finding a *simultaneous embedding with fixed edges* of a pair graphs. In particular, the results of this chapter can be used to solve this problem for a restricted class of inputs.

Problem extensions. Several generalizations of the PARTIALLYEMBEDDEDPLANARITY problem naturally arise. For the following two we readily conclude that they are NP-

complete (since they contain as special cases Crossing number and Maximum planar subgraph, respectively): (i) deciding if an embedding $\mathcal{H}$ can be extended to a planar drawing of G with at most k crossings; and (ii) deciding if at least k edges of $E(G) \setminus E(H)$ can be added to $\mathcal{H}$ preserving planarity.

Two additional problems that generalize Pep in different directions are the following: (iii) deciding whether G has a planar embedding $\mathcal{G}$ in which at least k edges of H are embedded the same as in $\mathcal{H}$; and (iv) deciding whether a set F of at most k edges can be deleted from H, so that $G \setminus F$ has a planar embedding $\mathcal{G}$ in which the induced embedding of $H \setminus F$ coincides with $\mathcal{H} \setminus F$. We show that even these two problems, called Minimum Rerouting Partially Embedded Planarity and Maximum Preserved Partially Embedded Planarity, respectively, are NP-hard.

Theorem 6.8. Minimum Rerouting Partially Embedded Planarity *and* Maximum Preserved Partially Embedded Planarity *are NP-hard.*

Proof. The proof is by reduction from SteinerTree in planar graphs, which is known to be NP-hard [GJ77]. Let $G = (V, E)$ be a planar graph and let $T \subseteq V$ be a set of terminals. Choose an embedding $\mathcal{G}$ of G and let H be the dual of G with embedding $\mathcal{H}$. For each terminal $t \in T$ we add a new vertex v_t to H and prescribe it inside the face $t^\star$ that is dual to t. Moreover let S be the edge set of any connected graph on the vertices v_t. We set $G' := H + S$.

Now consider the problem of identifying a set F of edges of H such that $G' - F$ can be drawn planar and such that the subgraph $H - F$ has embedding $\mathcal{H} - F$. Clearly S can be drawn if and only if we remove edges of H such that all vertices v_t lie in the same face. This is equivalent to the property that the set $F^\star$ of edges dual to F is a Steiner Tree in G with terminal set T.

This shows that Maximum Preserved Partially Embedded Planarity is NP-hard. Moreover, as the vertices v_t form a separate connected component, we can reinsert the edges of F without crossings into the drawing, i.e., it is sufficient to reroute the edges in F. This shows that Minimum Rerouting Partially Embedded Planarity is NP-hard as well. □

In the case of Maximum Preserved Partially Embedded Planarity we can even make H connected as follows. We connect each vertex v_t to an arbitrary vertex of its prescribed face and for S we choose the edge set of a star graph on the vertices v_t. Thus, Maximum Preserved Partially Embedded Planarity is NP-hard even if the prescribed graph H is connected. This does, however, not hold for Minimum Rerouting Partially Embedded Planarity, as the reduction relies on the property that every edge of each face can be removed and reinserted after drawing S. This is not the case if H is connected. We leave open the question whether Minimum Rerouting Partially Embedded Planarity is NP-hard if the prescribed graph H is connected.

Application to simultaneous embedding with fixed edges. Moreover, the results presented in this chapter can be used to solve special cases of the problem simultaneous embedding with fixed edges, Sefe for short, to which we also contribute in Chapter 8 of this thesis. A *simultaneous embedding with fixed edges* (in the following called Sefe, for short) of a pair $(G_1 = (V, E_1), G_2 = (V, E_2)$ of graphs on the same vertex set is a pair (Γ_1, Γ_2) of drawings such that: (i) Γ_i is a planar drawing of G_i, for each $i = 1, 2$; (ii) each vertex $v \in V$ is drawn on the same point in Γ_i, for every $i = 1, 2$; (iii) each edge

$(u, v) \in E_1 \cap E_2$ is represented by the same Jordan curve in Γ_1 and in Γ_2. The problem can also be generalized to simultaneous embedding of multiple graphs.

The SEFE problem is a well-studied problem in graph drawing. A lot of research has been devoted to pairs of graph classes that always admit a SEFE and to find out, how many bends are necessary in this case [EK04, DL07a, Fra06, FJKS08]. Additionally, a lot of work is concerned with the algorithmic aspects of the SEFE problem. In particular, it is known that SEFE is NP-hard for two geometric graphs, where the edges are restricted to be straight-line segments [EBGJ$^+$07a] and that general SEFE is NP-hard for three (or more) graphs [GJP$^+$06]. There also exist polynomial-time algorithms, but until now only for rather restricted cases, such as when the union of the graphs is homeomorphic to K_5 or to $K_{3,3}$ [FJKS08], or if one of the two graphs has at most one cycle [FGJ$^+$08b]. Despite this large amount of research, the question about the complexity status of the SEFE problem remains open.

The results presented in this chapter yield to solve in linear time the problem of deciding whether two graphs admit a simultaneous embedding with fixed edges, if one of the graphs has a fixed embedding. Jünger and Schulz [JS09] showed that two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ admit a SEFE if and only if they admit embeddings $\mathcal{E}_1$ and $\mathcal{E}_2$ of G_1 and G_2, respectively, that coincide on the intersection graph. As a consequence of the results we presented on the PEP problem, deciding whether two graphs have a SEFE is a linear-time solvable problem if one the graphs has a fixed embedding.

Theorem 6.9. *Let G_1 and G_2 be two graphs with the same n vertices, let $\mathcal{G}_2$ be a planar embedding of G_2 and let $\mathcal{G}_{1\cap2}$ be the restriction of $\mathcal{G}_2$ to $G_1 \cap G_2$. Then G_1 and G_2 have a* SEFE *in which the planar embedding of G_2 is $\mathcal{G}_2$ if and only if $(G_1, G_1 \cap G_2, \mathcal{G}_{1\cap2})$ is a* YES*-instance of* PEP*.*

More generally, the SEFE problem is linear-time solvable, if the embedding of the intersection graph $G_{1\cap2} := G_1 \cap G_2$ is fixed to an embedding $\mathcal{G}_{1\cap2}$. In this case, (G_1, G_2) is a YES-instance of SEFE if and only if $(G_1, G_{1\cap2}, \mathcal{G}_{1\cap2})$ and $(G_2, G_{1\cap2}, \mathcal{G}_{1\cap2})$ are both YES-instances of PEP. In particular, this contains all the cases where one of the graphs is triconnected, or if the intersection graph consists of a single triconnected component and possibly some isolated vertices, as in these cases the embedding of the intersection graph is fixed. In Chapter 8, we will have a closer look at the SEFE problem. There, we give an efficient algorithm for solving the case where the intersection graph $G_{1\cap2}$ is biconnected, and further consider the case where it is only connected, but not necessarily biconnected.

6.6. Concluding Remarks

In this chapter we have shown that PARTIALLYEMBEDDEDPLANARITY (PEP), that is, the problem of deciding whether a partial drawing can be extended to a planar drawing of the entire graph, is solvable in linear time. To derive this result, we first presented a combinatorial characterization of planar PEGs in terms of conditions on the structure of the triconnected, biconnected, and connected components of the input graph. This characterization immediately implies a polynomial-time algorithm for testing the planarity of a given PEG. The second part of the chapter is devoted to a careful implementation of the algorithm following from the characterization, resulting in an algorithm for PEP with optimal linear running time. While edge-compatibility exhibits a very local behavior, and is hence not too difficult to enforce in linear time, quite some steps are necessary to

handle cycle-compatibility as well. Finally, we showed that our testing algorithm can also be made constructive, that is, it finds an embedding extension for the input PEG, if one exists. Altogether, from a purely algorithmic point of view, this completely settles the problem PEP.

We further considered related problems and showed that several generalizations of PEP are NP-hard and showed that there are connections to another well-known graph drawing problem, the SEFE problem. The results in this chapter immediately imply a linear-time algorithm for solving SEFE when the embedding of the intersection graph is fixed, which holds for example if one of the input graphs or the intersection is triconnected. We will further study this problem in Chapter 8.

In light of the connection to the SEFE problem, for which we still lack a fundamental understanding, it seems to be important to understand the structure of the PEGs that admit an embedding and also the ones that do not. A systematic understanding of the structure of these PEGs might help to reduce the search space of the SEFE problem, as it may allow to find necessary conditions on the embedding of the intersection graph of a SEFE instance, and hence to reject a large portion of the possible embeddings of the intersection graph. Another motivation to further study the structure of planar PEGs is the objective to construct a *certifying algorithm*. A certifying algorithm is an algorithm that not just outputs the solution to a problem, but also a certificate for the correctness of the solution, which a user can then verify, usually using a much simpler procedure than the algorithm itself.

One prominent example of certifying algorithms are planarity tests. For the case that the input graph is planar, we expect the algorithm to also find a planar embedding, that is a rotation scheme of the vertices corresponding to a planar embedding. Testing whether such a rotation scheme actually corresponds to a planar embedding is much simpler, it is even possible to compute the genus of such an embedding efficiently by traversing the boundaries of all faces in order to compute their total number. Afterwards, the generalized Euler formula $n - m + f = 2 - 2g$ allows the computation of the genus. An embedding is planar if and only if its genus is 0. For the negative case, where a graph is non-planar, these algorithms usually find a subdivision of K_5 or $K_{3,3}$ as a certificate of non-planarity. Again, given a subgraph of the input graph, it is not difficult to check that it actually is a subdivision of a Kuratowski graph. Thus, the correctness of the outputs of a certifying algorithm can easily be checked.

In particular, for very complicated algorithms, such as many planarity tests, this is very helpful. Indeed the shortcomings of the first implementations of linear-time planarity tests were one of the main motivations to construct certifying algorithms, to enable users to at least check whether the decision of the algorithm was correct [MMNS10]. In the next chapter we study the planarity of PEGs further in this sense, and propose a way to construct such certificates of non-planarity, by characterizing the class of planar PEGs in terms of forbidden substructures, similar to the Kuratowski theorem, which characterizes the class of planar graphs by the forbidden minors $K_{3,3}$ and K_5. The ultimate goal is of course a certifying algorithm that for a given PEG $(G, H, \mathcal{H})$ either outputs an embedding extension, if one exists, or finds a forbidden substructure that enables the user to check that an embedding extension does not exist.

Much unlike the previous chapters, this chapter has revealed a vastly different face of planarity. Planarity is a strong property and SPQR-trees allow for a succinct representation of all planar embeddings of biconnected planar graphs. This helps a lot to make embedding problems of planar graphs tractable. Nevertheless, it is quite surprising that PEP is even

linear-time solvable, although typically problems asking to extend a given partial solution are much more difficult than their counterpart without a partial input. In particular, the result that the choice of embedding can be performed independently for each skeleton of the SPQR-tree to yield a global solution, enables us to add the constraint that a subgraph has to have a certain embedding as an afterthought to many other embedding problems using the SPQR-tree. For example, in Chapter 9 we consider the problem of finding an orthogonal drawing with at most one bend per edge. The algorithm essentially performs dynamic programming on the SPQR-tree. By restricting the consideration to compatible embeddings of the skeletons, one can easily enforce a certain combinatorial embedding on a subgraph.

Open Problems. The problem Pep asks for the extendability of combinatorial planar embeddings, which corresponds to topological drawings of planar graphs. An obvious question is to consider other drawing styles. It is known that completing partial straight-line drawings is NP-hard [Pat06], and it seems that the NP-hardness proof easily generalizes to poly-line drawings that admit a fixed number of bends per edge. However, it may be interesting to consider the problems of extending for example orthogonal drawings, or drawings in the Manhattan-geodesic style, as introduced by Katz et al. [KKRW10].

Another way to generalize Pep is to relax the strict condition that the subgraph H has to have a fixed embeddings. Mutzel et al. [GKM08] consider embedding constraints that are expressed in terms of PQ-trees, restricting the orders of the incident edges of each vertex. The common generalization of Pep and this problem assumes that only a subgraph is constrained by such PQ-trees and the remaining edges can be inserted arbitrarily. Is it possible to decide planarity of a partially PQ-constrained graph in polynomial time?

Chapter 7

A Kuratowski-Type Theorem for Planarity of Partially Embedded Graphs

In this chapter, we complement the work on planarity of partially embedded graphs from the previous chapter with a characterization of planar partially embedded graphs via forbidden substructures.

To this end, we introduce a containment relation of PEGs analogous to graph minor containment, and characterize the minimal non-planar PEGs with respect to this relation. We show that all the minimal non-planar PEGs except for finitely many belong to a single easily recognizable and explicitly described infinite family. We also describe a more complicated containment relation that only has a finite number of minimal non-planar PEGs. Furthermore, by extending the planarity test for PEGs presented in Chapter 6, we obtain a polynomial-time algorithm that, for a given PEG, either produces a planar embedding or identifies a minimal obstruction, that is, a certificate of non-planarity.

The chapter is based on joint work with Vít Jelínek and Jan Kratochvíl [JKR11].

7.1. Introduction

Recall that a partially embedded graph (PEG) is a triplet $(G, H, \mathcal{H})$, where G is a graph, H is a subgraph of G, and $\mathcal{H}$ is a planar embedding of H. The problem PARTIALLYEMBEDDEDPLANARITY (PEP) asks whether a PEG $(G, H, \mathcal{H})$ admits a planar (non-crossing) embedding of G whose restriction to H is $\mathcal{H}$. In this case we say that the PEG $(G, H, \mathcal{H})$ is planar. Despite of this being a very natural generalization of planarity, this approach has not been considered until recently. It should be mentioned that all previous planarity testing algorithms have been of little use for PEP, as they all allow flipping of already drawn parts of the graph, and thus are not suitable for preserving an embedding of a given subgraph.

In this chapter we complement the algorithm from the previous chapter by a study of the combinatorial aspects of the question which PEGs are planar. In particular, we provide a complete characterization of planar PEGs via a small set of forbidden substructures, similar to the celebrated Kuratowski theorem [Kur30] that characterizes planarity via the forbidden minors K_5 and $K_{3,3}$. Our characterization can then be used to modify the

existing planarity test for partially embedded graphs into a certifying algorithm that either finds a solution or finds a certificate, that is, a forbidden substructure, that shows that the instance is not planar.

As we have already seen in the conclusion of Chapter 6, the planarity of PEGs also has some connections to the problem *simultaneous embedding with fixed edges*, or SEFE for short, which also is the topic of Chapter 8. It asks whether two graphs G_1 and G_2 on the same vertex set V admit two drawings Γ_1 and Γ_2 of G_1 and G_2, respectively, such that (i) all vertices are mapped to the same point in Γ_1 and Γ_2, (ii) each drawing Γ_i is a planar drawing of G_i for $i = 1, 2$, and (iii) edges common to G_1 and G_2 are represented by the same Jordan curve in Γ_1 and Γ_2. Jünger and Schulz [JS09] show that two graphs admit a SEFE if and only if they admit planar embeddings that coincide on the intersection graph. Once an embedding $\mathcal{H}$ of the intersection graph H has been fixed, it remains to check whether the PEGs $(G_1, H, \mathcal{H})$ and $(G_2, H, \mathcal{H})$ are both planar. Understanding the forbidden substructures for planarity of PEGs may be particularly beneficial in studying this problem since our obstructions give an understanding of which configurations should be avoided when searching for a feasible embedding of the intersection graph.

For the purposes of our characterization, we introduce a set of operations that preserve the planarity of PEGs. Note that it is not possible to use the usual minor operations, as sometimes, when contracting an edge of G not belonging to H, it is not clear how to modify the embedding of H. Our minor-like operations are defined in Section 7.2.

Our goal is to identify all minimal non-planar PEGs in the minor-like order determined by our operations; such PEGs are referred to as *obstructions*. Our main theorem says that all obstructions are depicted in Figure 7.1 or belong to a well described infinite class of so-called *alternating chains* (the somewhat technical definition is postponed to Section 7.2). It can be verified that each of them is indeed a minimal obstruction, that is, it is not planar, but applying any of the PEG-minor operations results in a planar PEG.

We say that a PEG *avoids* a PEG X if it does not contain X as a PEG-minor. Furthermore, we say that a PEG is *obstruction-free* if it avoids all PEGs of Figure 7.1 and all alternating chains of lengths $k \geq 4$. Then our main theorem can be expressed as follows.

Theorem 7.1. *A* PEG *is planar if and only if it is obstruction-free.*

Since our PEG-minor operations preserve planarity, and since all the listed obstructions are non-planar, any planar PEG is obstruction-free. The main task is to prove that an obstruction-free PEG is planar.

Having identified the obstructions, a natural question is whether the PEG-planarity testing algorithm of Chapter 6 can be extended so that it provides an obstruction if the input is non-planar. This is indeed so.

Theorem 7.2. *There is a polynomial-time algorithm that for an input* PEG $(G, H, \mathcal{H})$ *either constructs a planar embedding of G extending $\mathcal{H}$, or provides a certificate of non-planarity, that is, identifies an obstruction present in $(G, H, \mathcal{H})$ as a* PEG*-minor.*

Outline. The chapter is organized as follows. In Section 7.2, we first recall some basic definitions and results on PEGs and their planarity, and then define the PEG-minor order and the alternating chain obstructions. In Section 7.3, we show that the main theorem holds for instances where G is biconnected. We extend the main theorem to general (not necessarily biconnected) PEGs in Section 7.4. In Section 7.5, we present possible strengthenings of our PEG-minor relations, and show that when more complicated reduction rules are allowed, the modified PEG-minor order has only finitely many non-planar PEGs.

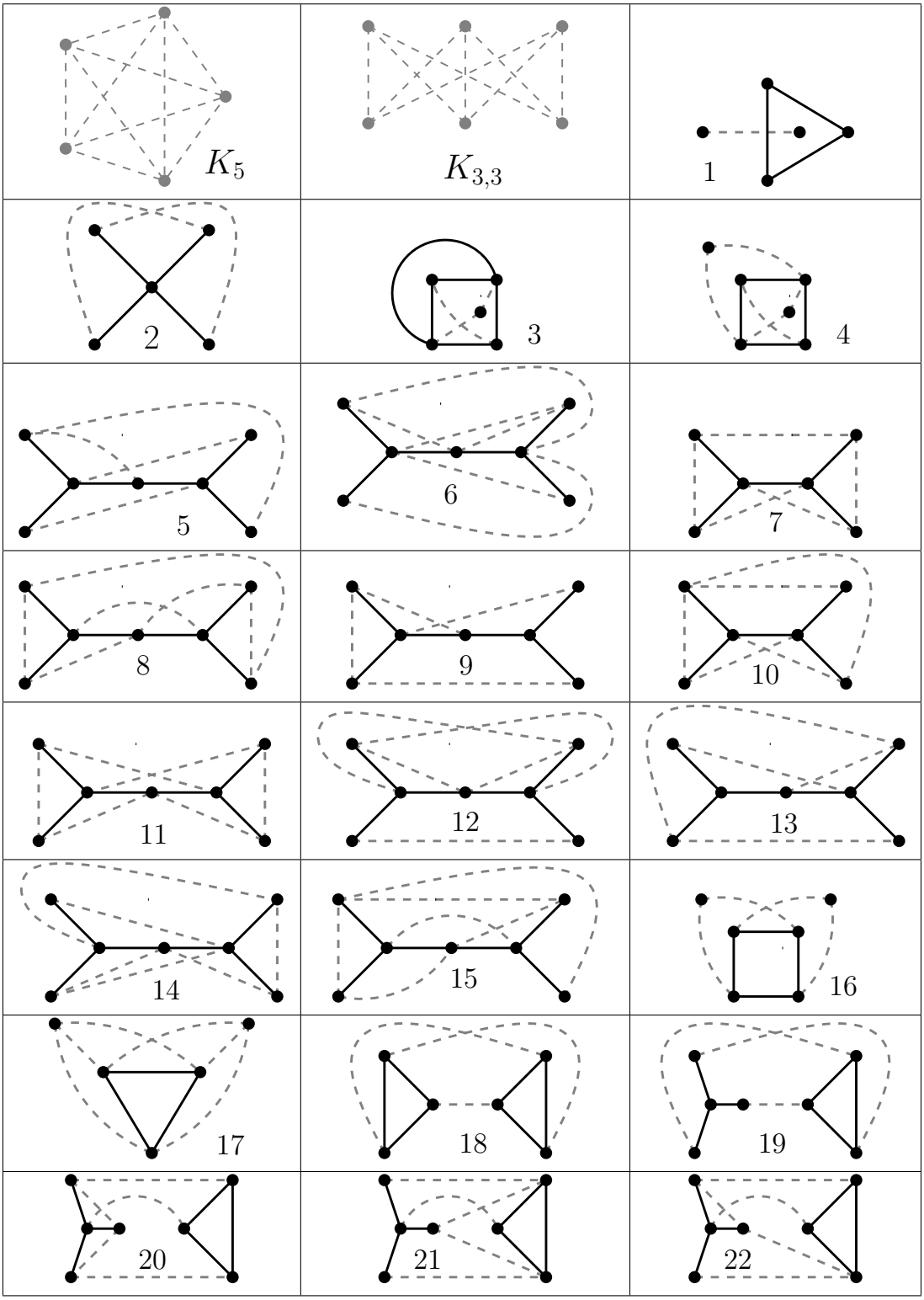


Figure 7.1.: The minimal obstructions not equal to the k-fold alternating chains for $k \geq 4$. The black solid edges belong to H, the light dashed edges to G, but not to H. All the vertices belong to both G and H, except for K_5 and $K_{3,3}$, where H is empty.

In Section 7.6 we briefly provide an argument for Theorem 7.2 and then conclude with some open problems.

7.2. Preliminaries and Notation

In this section we introduce basic definitions that we use throughout this chapter. Since the graphs we consider in this chapter are not necessarily connected, we again use the more general notion of an embedding on the sphere, where an embedding is described by the rotation scheme and the face boundaries. As in the combinatorial part of Chapter 6, we use the unrooted version of the SPQR-tree for handling biconnected PEGs. We further assume that the reader is familiar with the basic concepts of partially embedded graphs, as introduced in Chapter 6.

Minor-like operations for partially embedded graphs. We first introduce a set of operations that preserve planarity when applied to a PEG $I = (G, H, \mathcal{H})$. The set of operations is chosen so that the resulting instance $I' = (G', H', \mathcal{H}')$ is again a PEG (in particular, the prescribed graph H' is a subgraph of G' and $\mathcal{H}'$ is a planar embedding of H'). It is not possible to use the usual minor operations, as sometimes, when contracting an edge of $G - H$, the embedding of the modified graph H is not unique and some of the possible embeddings lead to planar PEGs, while some do not.

We will consider seven minor-like operations, of which the first five are straightforward.

1. Vertex removal: Remove from G and H a vertex $v \in V(G)$ with all its incident edges.
2. Edge removal: Remove from G and H an edge $e \in E(G)$.
3. Vertex relaxation: For a vertex $v \in H$ remove v and all its incident edges from H, but keep them in G. In other words, vertex v no longer has a prescribed embedding.
4. Edge relaxation: Remove an edge $e \in E(H)$ from H, but keep it in G.
5. H-edge contraction: Contract an edge $e \in E(H)$ in both G and H, update $\mathcal{H}$ accordingly.

The contraction of G-edges is tricky, as we have to care about two things. First, we have to take care that the embedding induced by $\mathcal{H}$ on the modified subgraph H' remains planar and second, even if it remains planar we do not want to create a new cycle C in H as in this case the relative positions of the connected components of H with respect to this cycle may not be uniquely determined. We therefore have special requirements for the G-edges that may be contracted and we distinguish two types, one of which trivially ensures the above two conditions and one that explicitly ensures them.

6. Simple G-edge contraction: Assume that $e = uv$ is an edge of G, such that at least one of the two vertices u and v does not belong to H. Contract e in G, and leave H and $\mathcal{H}$ unchanged.
7. Complicated G-edge contraction: Assume that $e = uv$ is an edge of G, such that u and v belong to distinct components of H, but share a common face of $\mathcal{H}$. Assume further that both u and v have degree at most 1 in H. This implies that we may uniquely extend $\mathcal{H}$ to an embedding $\mathcal{H}^+$ of the graph H^+ that is obtained from H by adding the edge uv. Afterwards we perform an H-edge contraction of the edge uv to obtain the new PEG.

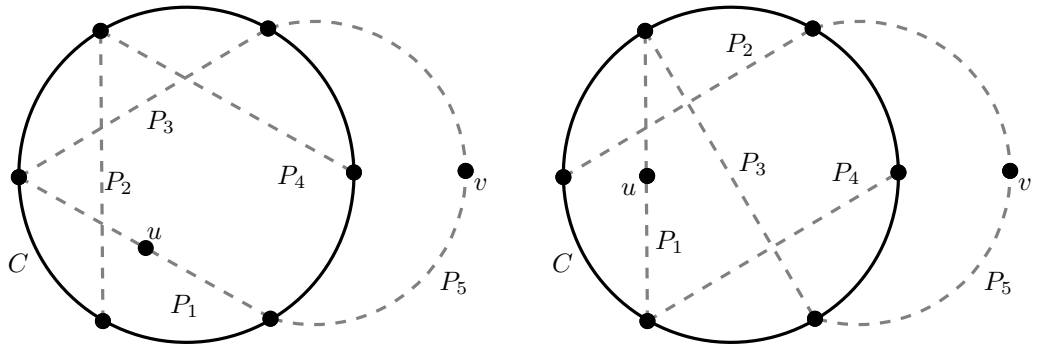


Figure 7.2.: Two non-isomorphic 5-fold alternating chains.

If a contraction produces multiple edges, we only preserve a single edge from each such set of multiple edges, so that G and H remain simple.

Note that the resulting embedding $\mathcal{H}$ may depend on which edge we decide to preserve.

The extra conditions in the G-edge contractions ensure that the embedding $\mathcal{H}$ of the modified graph H is uniquely determined from the initial embedding of H. The conditions on vertex degrees in H ensure that the rotation scheme of the H-edges around the resulting vertex is unique. In the complicated G-edge contraction, the requirement that the endpoints need to lie in distinct connected components of H that share a face ensures that the contraction does not create a new cycle in H and that the resulting graph H has a unique planar embedding induced by $\mathcal{H}$.

Let $(G, H, \mathcal{H})$ be a PEG and let $(G', H', \mathcal{H}')$ be the result of one of the above operations on $(G, H, \mathcal{H})$. It is not hard to see that an embedding $\mathcal{G}$ of G that extends $\mathcal{H}$ can be transformed into an embedding $\mathcal{G}'$ of G' that extends $\mathcal{H}'$, as in fact all of them can be expressed as usual minor operations on G. Therefore, all the above operations preserve planarity of PEGs. If a PEG A can be obtained from a PEG B by applying a sequence of the above operations, we say that A is a PEG-*minor* of B or that B *contains* A *as a* PEG-*minor*.

Alternating chains. Apart from the obstructions in Figure 7.1, there is an infinite family of obstructions, which we call *alternating chains*. To describe them, we need some terminology. Let C be a cycle of length at least four, and let u, v, x and y be four distinct vertices of C. We say that the pair of vertices $\{u, v\}$ *alternates* with the pair $\{x, y\}$ on C, if u and v belong to distinct components of $C - x - y$.

Intuitively, an alternating chain consists of a prescribed cycle C and a sequence of internally disjoint paths $P_1, \ldots, P_k$ of which only the endpoints belong to C, in such a way that the endpoints of P_i alternate exactly with the endpoints of P_{i+1} on C for $i = 1, \ldots, k-1$. Now assume that P_1 contains a vertex that is prescribed inside C. Due to the fact that the endpoints of consecutive paths alternate this implies that all P_i with i odd must be embedded inside C, while all P_i with i even must be embedded outside. A k-fold alternating chain is such that the last path P_k is prescribed in a way that contradicts this, that is, it is prescribed inside C if k is even and outside, if k is odd. Generally it is sufficient to have paths of length 1 for $P_2, \ldots, P_{k-1}$ and to have a single vertex (for the prescription) in each of P_1 and P_k. We now give a precise definition.

Let $k \geq 3$ be an integer. A *k-fold alternating chain* is a PEG $(G, H, \mathcal{H})$ of the following form:

- The graph H consists of a cycle C of length $k+1$ and two isolated vertices u and v.

If k is odd, then u and v are embedded on opposite sides of C in $\mathcal{H}$, otherwise they are embedded on the same side.

- The graph G has the same vertex set as H, and the edges of G that do not belong to H form k edge-disjoint paths $P_1, \ldots, P_k$, whose endpoints belong to C. The path P_1 has two edges and contains u as its middle vertex, the path P_k has two edges and contains v as its middle vertex, and all the other paths have only one edge.
- The endpoints of the path P_i alternate with the endpoints of the path P_j on C if and only if $j = i + 1$ or $i = j + 1$.
- All the vertices of C have degree 4 in G (that is, each of them is a common endpoint of two of the paths P_i), with the exception of two vertices of C that have degree three. One of these two vertices is an endpoint of P_2, and the other is an endpoint of P_{k-1}.

Let Ach_k denote the set of k-fold alternating chains. It can be checked that for each $k \geq 4$, the elements of Ach_k are minimal obstructions; we will prove this in Lemma 7.15. Obstruction 4 from Figure 7.1 is actually the unique member of Ach_3, and is a minimal obstruction as well. However, we prefer to present it separately as an 'exceptional' obstruction, because we often need to refer to it explicitly. Note that for $k \geq 5$ we may have more than one non-isomorphic k-fold chain; see Figure 7.2.

7.3. Biconnected Pegs

In this section we prove Theorem 7.1 for biconnected Pegs. We first recall a characterization of biconnected planar Pegs via SPQR-trees from Chapter 6.

Definition 7.1 (Definitions 6.1, 6.2, and 6.3 of Chapter 6.)**.** Let $(G, H, \mathcal{H})$ be a biconnected Peg. A planar embedding of the skeleton of a node of the SPQR-tree of G is *edge-compatible with* $\mathcal{H}$ if, for every vertex x of the skeleton and for every three edges of H incident to x that project to different edges of the skeleton, their order determined by the embedding of the skeleton is the same as their order around x in $\mathcal{H}$.

A planar embedding of the skeleton $\mathfrak{S}$ of a node μ of the SPQR-tree of G is *cycle-compatible with* $\mathcal{H}$ if, for every facial cycle $\vec{C}$ of $\mathcal{H}$ whose edges project to a simple cycle $\vec{C}'$ in $\mathfrak{S}$, all the vertices of $\mathfrak{S}$ that lie to the left of $\vec{C}$ and all the skeleton edges that contain vertices that lie to the left of $\vec{C}$ in $\mathcal{H}$ are embedded to the left of $\vec{C}'$; and analogously for the vertices to the right of $\vec{C}$.

A planar embedding of a skeleton of a node of the SPQR-tree of G is *compatible* if it is both edge- and cycle-compatible.

In Chapter 6 we have shown that a biconnected Peg is planar if and only if the skeleton of each node admits a compatible embedding; see Theorem 6.1. We use this characterization and show that any skeleton of a biconnected Peg that avoids all obstructions admits a compatible embedding. Since S-nodes have only one embedding, and their embedding is always compatible, we consider P- and R-nodes only. The two types of nodes are handled separately in Subsections 7.3.1 and 7.3.2, respectively.

The following lemma will be useful in several parts of the proof.

Lemma 7.1. *Let $(G, H, \mathcal{H})$ be a Peg, let u be a vertex of a skeleton $\mathfrak{S}$ of a node μ of the SPQR-tree of G, and let e be an edge of $\mathfrak{S}$ with endpoints u and v. Let $F \subseteq E(H)$ be the*

set of edges of H that are incident to u and project into e. If the edges of F do not form an interval in the rotation scheme of u in $\mathcal{H}$ then $(G, H, \mathcal{H})$ contains obstruction 2.

Proof. If F is not an interval in the rotation scheme, then there exist edges $f, f' \in F$ and $g, g' \in E(H) \setminus F$, all incident to u, and appearing in the cyclic order f, g, f', g' around u in $\mathcal{H}$. Let x and x' be the endpoints of f and f' different from u and let y and y' be the endpoints of g and g' different from u. For a skeleton edge e, we let G_e be the expansion graph of e.

If μ is an S-node, then g and g' project to the same skeleton edge uw with $v \neq w$. Note that G_{uv} and G_{uw} share only the vertex u and moreover, they are both connected even after removing u. Therefore, there exist disjoint paths P in G_{uv} and Q in G_{uw} connecting x to x' and y to y', respectively. We may relax all internal vertices and all edges of P and Q, and then perform simple edge contractions to replace each of the two paths with a single edge. This yields obstruction 2.

If μ is an R-node, then $G_{uv} - u$ is connected, and hence it contains a path P from x to x'. Moreover, since $G - G_{uv}$ is connected, it has a path Q from y to y'. As in the previous case, contraction of P and Q yields obstruction 2.

If μ is a P-node, then $G_e - \{u, v\}$ is connected, and therefore there is a path P connecting x to x' in $G_e - \{u, v\}$. Analogous to the previous cases, a path Q from y to y' exists that avoids u and P. Again their contraction yields obstruction 2. □

In the following, we assume that the H-edges around each vertex of a skeleton that project to the same skeleton edge form an interval in the rotation scheme of this vertex.

7.3.1. P-Nodes

Throughout this section, we assume that $(G, H, \mathcal{H})$ is a biconnected obstruction-free PEG. We fix a P-node μ of the SPQR-tree of G, and we let $\mathfrak{P}$ be its skeleton. Let u and v be the two vertices of $\mathfrak{P}$, and let $e_1, \ldots, e_k$ be its edges. Let G_i be the pertinent graph of e_i. As we know, each graph G_i is either a single edge connecting u and v, or it does not contain the edge uv and $G_i - \{u, v\}$ is connected (otherwise the SPQR-tree would have two adjacent P-nodes which could be simplified into a single P-node).

The goal of this section is to prove that $\mathfrak{P}$ admits a compatible embedding. We first deal with edge-compatibility.

Lemma 7.2. *The P-skeleton $\mathfrak{P}$ has an edge-compatible embedding.*

Proof. If $\mathfrak{P}$ has no edge-compatible embedding, then the rotation scheme around u conflicts with the rotation scheme around v. This implies that there is a triplet of skeleton edges e_a, e_b, e_c, for which the rotation scheme around u imposes a different cyclic order than the rotation scheme around v. We distinguish two cases.

Case 1. The graph H has a cycle C whose edges intersect two of the three skeleton edges, say e_a and e_b. Then the edge e_c must contain a vertex x whose prescribed embedding is to the left of C, as well as a vertex y whose prescribed embedding is to the right of C. To see this, note that the expansion graph of e_c cannot be a single edge, as this would imply that H is not embedded in a planar way. Hence x and y can be taken as the other endpoint of H-edges belonging to e_c, incident to u and v, respectively. Since x and y are connected by a path in $G_c - \{u, v\}$, we obtain obstruction 1.

Case 2. The graph H has no cycle that intersects two of the three $\mathfrak{P}$-edges e_a, e_b, e_c. Each of the three $\mathfrak{P}$-edges contains an edge of H adjacent to u as well as an edge of H

adjacent to v. Since $G_i - \{u, v\}$ is connected for each i, it follows that each of the three skeleton edges contains a path from u to v, such that the first and the last edge of the path belong to H. Fix such paths P_a, P_b and P_c, projecting into e_a, e_b and e_c, respectively.

Moreover, at least two of these paths (P_a and P_b, say) also contain an edge not belonging to H, otherwise they would form a cycle of H intersecting two skeleton edges. By relaxations and simple contractions, we may reduce P_a to a path of length three, whose first and last edge belong to $E(H)$ and the middle edge belongs to $E(G) \setminus E(H)$. The same reduction can be performed with P_b. The path P_c can then be contracted to a single vertex, to obtain obstruction 2. □

Next, we consider cycle-compatibility. Assume that H has at least one facial cycle whose edges intersect two distinct skeleton edges. It follows that u and v belong to the same connected component of H; denote this component by H_{uv}. We call *uv-cycle* any facial cycle of H that contains both u and v. Note that any uv-cycle is also a facial cycle of H_{uv}, and a facial cycle of H_{uv} that contains both u and v is a uv-cycle. Following the convention of Chapter 6, we assume that all facial cycles are oriented in such a way that a face is to the left of its facial cycles. The next lemma shows that the vertices of H_{uv} cannot violate any cycle-compatibility constraints without violating edge-compatibility as well.

Lemma 7.3. *Assume that C is a uv-cycle that intersects two distinct $\mathfrak{P}$-edges e_a and e_b, and that x is a vertex of H_{uv} not belonging to C. In any edge-compatible embedding of $\mathfrak{P}$, the vertex x does not violate cycle-compatibility with respect to C.*

Proof. The vertex x belongs to a skeleton edge e_x different from e_a and e_b, otherwise it cannot violate cycle-compatibility. Note that since x is in H_{uv}, e_x must contain a path P of H that connects x to one of the poles u and v. In the graph H, all the vertices of P must be embedded on the same side of C as the vertex x. The last edge of P may not violate edge-compatibility, which forces the whole edge e_x, and thus x, to be embedded on the correct side of the projection of C, as claimed. □

The next lemma shows that for an obstruction-free Peg, all vertices of H projecting to the same $\mathfrak{P}$-edge impose the same cycle-compatibility constraints for the placement of this edge. The basic idea of the proof is that two vertices x and y projecting into the same skeleton edge must be connected by a G-path that projects into this edge as well. Therefore, if x and y had different positions with respect to a cycle of $\mathcal{H}$, we would obtain obstruction 1.

Lemma 7.4. *Let x and y be two vertices of H, both distinct from u and v. Suppose that x and y belong to the same $\mathfrak{P}$-edge e_a. Let C be a cycle of H that is edge-disjoint from G_a. Then x and y are embedded on the same side of C in $\mathcal{H}$.*

Proof. Since $G_a - \{u, v\}$ is a connected subgraph of G, there is a path P in G that connects x to y and avoids u and v. Since C is edge-disjoint from G_a, the path P avoids all the vertices of C. If x and y were not embedded on the same side of C, we would obtain obstruction 1 by contracting C and P. □

We are now ready to prove the main result of this subsection.

Proposition 7.1. *Let $(G, H, \mathcal{H})$ be a biconnected obstruction-free* Peg. *Then every P-skeleton $\mathfrak{P}$ of the SPQR-tree of G admits a compatible embedding.*

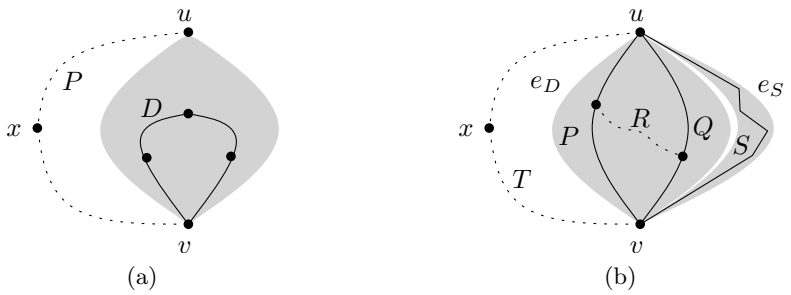


Figure 7.3.: Illustrations of Case 1 (a) and Case 2.a (b) in the proof of Proposition 7.1. The relevant edges of the skeleton are shown as shaded regions.

Proof. Fix an edge-compatible embedding that minimizes the number of violated cycle-compatibility constraints; more precisely, fix an embedding of $\mathfrak{P}$ that minimizes the number of pairs (C, x) where C is a facial cycle of $\mathcal{H}$ projecting to a cycle C' of $\mathfrak{P}$, x is a vertex of $H - \{u, v\}$ projecting into a skeleton edge e_x not belonging to C', and the relative position of C' and e_x in the embedding of $\mathfrak{P}$ is different from the relative position of C and x in $\mathcal{H}$. We claim that the chosen embedding of $\mathfrak{P}$ is compatible.

For contradiction, assume that there is at least one pair (C, x) that violates cycle-compatibility in the sense described above. Let e_x be the $\mathfrak{P}$-edge containing x. Note that e_x does not contain any edge of H adjacent to u or v. If it contained such an edge, it would contain a vertex y from the component H_{uv}, and this would contradict Lemma 7.3 or Lemma 7.4. Thus, the edge e_x does not participate in any edge-compatibility constraints.

It follows that x does not belong to the component H_{uv}. That means that in H, the vertex x is embedded in the interior of a unique face F of H_{uv}. We distinguish two cases, depending on whether the boundary of F contains both poles u and v of $\mathfrak{P}$ or not.

Case 1. The boundary of F contains at most one of the two poles u and v. Without loss of generality, the boundary does not contain u. Thus, F has a facial cycle D that separates u from x. The pertinent graph G_x of e_x contains a path P from x to u that avoids v. The path P does not contain any vertex of H_{uv} except u, and in particular, it does not contain any vertex of D; see Fig 7.3a. Contracting D to a triangle and P to an edge yields obstruction 1, which is a contradiction.

Case 2. The boundary of F contains both poles u and v of the skeleton. In this case, since u and v belong to the same block of H, the face F has a unique facial cycle D that contains both u and v. The cycle D is the only uv-cycle that has x to its left (that is, inside its corresponding face).

The cycle D may be expressed as a union of two paths P and Q connecting u and v, where P is directed from u to v and Q is directed from v to u. We distinguish two subcases, depending on whether the paths P and Q project to different $\mathfrak{P}$-edges.

Case 2.a Both P and Q project to the same skeleton edge e_D. Then each of the two paths has at least one internal vertex. Since all these internal vertices are inside a single skeleton edge, there must be a path R in G connecting an internal vertex of P to an internal vertex of Q and avoiding both u and v. By choosing R as short as possible, we may assume that no internal vertex of R belongs to D. Furthermore, since $\mathfrak{P}$ has at least one cycle-compatibility constraint, it must contain at least two edges that contain an H-path from u to v. In particular, there must exist a $\mathfrak{P}$-edge e_S different from e_D that contains an H-path S from u to v. Necessarily, the path S is embedded outside the face F, that is, to the right of D. And finally, the edge e_x must contain a G-path T from u to v that

contains x. Note that e_x is different from e_D and e_S, because e_x has no H-edge incident to u or v. Thus, the paths P, Q, S, T are all internally disjoint; see Figure 7.3b. The five paths P, Q, R, S, and T can then be contracted to form obstruction 3.

Case 2.b The two paths P and Q belong to distinct skeleton edges e_P and e_Q. That means that the facial cycle D projects to a cycle D' of the skeleton, formed by the two edges. Modify the embedding of the skeleton by moving e_x so that it is to the left of D'. This change does not violate edge-compatibility, because e_x has no H-edge adjacent to u or v.

We claim that in the new skeleton embedding, x does not participate in any violated cycle-compatibility constraint. To see this, we need to check that x is embedded to the right of any facial cycle $B \neq D$ of H_{uv} that projects to a cycle in the skeleton. Choose such a cycle B and let B' be its projection. Let e^+ or e^- denote the edges of D incident to u with e^+ being oriented towards u and e^- out of u. Similarly, let f^+ and f^- be the incoming and outgoing edges of B adjacent to u. In H, the four edges must visit u in the clockwise order (e^+, e^-, f^+, f^-), with the possibility that $e^- = f^+$ and $e^+ = f^-$.

Since the embedding of the skeleton is edge-compatible, this means that any skeleton edge embedded to the left of D' is also to the right of B', as needed. We conclude that in the new embedding of $\mathfrak{P}$, the vertex x does not violate any cycle-compatibility constraint, and by Lemma 7.4, the same is true for all the other H-vertices in e_x. Moreover, the change of embedding of e_x does not affect cycle-compatibility of vertices not belonging to e_x, so the new embedding violates fewer cycle-compatibility constraints than the old one, which is a contradiction. This proves that $\mathfrak{P}$ has a compatible embedding. □

7.3.2. R-Nodes

Let us now turn to the analysis of R-nodes. As in the case of P-nodes, our goal is to show that if a skeleton $\mathfrak{R}$ of an R-node in the SPQR-tree of G has no compatible embedding, then the corresponding PEG $(G, H, \mathcal{H})$ contains an obstruction. The skeletons of R-nodes have a more complicated structure than the skeletons of P-nodes, and accordingly, our analysis is more complicated as well. Similar to the case of P-nodes, we will first show that an R-node of an obstruction-free PEG must have an edge-compatible embedding, and as a second step show that in fact it must also have an edge-compatible embedding that also is cycle-compatible. Unfortunately, both steps are considerably more complicated than in the P-node case.

A skeleton of an R-node is a 3-connected graph. We therefore start with some preliminary observations about 3-connected graphs, which will be used throughout this section. Let $\mathfrak{R}$ be a 3-connected graph with a planar embedding $\mathfrak{R}^+$, let x be a vertex of $\mathfrak{R}$. A vertex y of $\mathfrak{R}$ is *visible from* x if $x \neq y$ and there is a face of $\mathfrak{R}^+$ containing x and y on its boundary. An edge e is visible from x if e is not incident with x and there is a face containing both x and e on its boundary. The vertices and edges visible from x form a cycle in $\mathfrak{R}$. To see this, note that these vertices and edges form a face boundary in $\mathfrak{R}^+ - x$, and every face boundary in a 2-connected graph is a cycle. We call this cycle *the horizon of* x.

In the following we will consider a fixed skeleton $\mathfrak{R}$ of an R-node. Since $\mathfrak{R}$ is 3-connected it has two planar embedding, denoted by $\mathfrak{R}^+$ and $\mathfrak{R}^-$. Suppose that neither of the two embeddings is compatible. The constraints on the embedding either stem from a vertices whose incident edges project to distinct edges of $\mathfrak{R}$ or from a cycle of $\mathfrak{R}$ that is a projection of an H-cycle whose cycle-compatibility constraints demand exactly one of the two embeddings. Since neither $\mathfrak{R}^+$ nor $\mathfrak{R}^-$ are compatible, there must be two of such

objects, one requiring embedding $\mathfrak{R}^+$, and the other one requiring $\mathfrak{R}^-$. If these objects are far apart in $\mathfrak{R}$, for example, if no vertex of the first object belongs to the horizon of a vertex of the second object, it is usually not too difficult to find one of the obstructions. However, if they are close together, a lot of special cases can occur. A significant part of the proof therefore consists in controlling the distance of objects and showing that either an obstruction is present or close objects cannot require different embeddings.

As before, we distinguish two main cases: first, we deal with the situation in which both embeddings of $\mathfrak{R}$ violate edge-compatibility. Next, we consider the situation in which $\mathfrak{R}$ has at least one edge-compatible embedding, but no edge-compatible embedding is cycle-compatible.

$\mathfrak{R}$ has no edge-compatible embedding

Let u be vertex of $\mathfrak{R}$ that violates the edge-compatibility of $\mathfrak{R}^+$, and let v be a vertex violating edge-compatibility of $\mathfrak{R}^-$. If $u = v$, that is, if a single vertex violates edge-compatibility in both embeddings, the following lemma shows that we can find an occurrence of obstruction 2 in $(G, H, \mathcal{H})$.

Lemma 7.5. *Assume that an R-node skeleton $\mathfrak{R}$ has a vertex u that violates edge-compatibility in both embeddings of $\mathfrak{R}$. Then $(G, H, \mathcal{H})$ contains obstruction 2.*

Proof. Let $B_1, \ldots, B_m$ be the $\mathfrak{R}$-edges incident to u that contain at least one H-edge incident to u. Assume that these edges are listed in their clockwise order around u in the embedding $\mathfrak{R}^+$. Let e_i be an H-edge incident with u projecting into B_i. By Lemma 7.1 if a triplet of edges B_i, B_j, B_k violates edge-compatibility in $\mathfrak{R}^+$, then this violation is demonstrated by the edges e_i, e_j, e_k.

Choose a largest set $I \subseteq \{1, \ldots, m\}$ such that the edges $\{e_i, i \in I\}$ do not contain any violation of edge-compatibility when embedded according to $\mathfrak{R}^+$. Clearly, $3 \leq |I| < m$. Choose an index $i \in \{1, \ldots, m\}$ not belonging to I. By maximality of I, there are distinct $j, k, \ell \in I$, different from i, such that (e_i, e_j, e_k, e_ℓ) appear clockwise in $\mathfrak{R}^+$ and (e_j, e_i, e_k, e_ℓ) appear clockwise in $\mathcal{H}$ (recall that (e_j, e_k, e_ℓ) have the same order in $\mathfrak{R}^+$ and $\mathcal{H}$, by the definition of I).

For $a \in \{1, \ldots, m\}$ let x_a be the endpoint of the skeleton edge B_a different from u. The horizon of u in $\mathfrak{R}^+$ contains two disjoint paths P and Q joining x_i with x_ℓ and x_j with x_k. By obvious contractions we obtain obstruction 2. □

Let us concentrate on the more difficult case when u and v are distinct. To handle this case, we introduce the concept of *wrung obstructions*. A wrung obstruction is a PEG $(G, H, \mathcal{H})$ with the following properties.

- G is a subdivision of a 3-connected planar graph, therefore it has two planar embeddings $\mathcal{G}^+$, $\mathcal{G}^-$.
- H has two distinct vertices u and v of degree 3. Any other vertex of H is adjacent to u or v, and any edge of H is incident to u or to v. Hence, H has five or six edges, and at most eight vertices.
- H is not isomorphic to $K_{2,3}$ or to K_4^- (that is, K_4 with an edge removed). Equivalently, H has at least one vertex of degree 1.
- The embedding $\mathcal{H}$ of H is such that its rotation scheme around u is consistent with $\mathcal{G}^+$ and its rotation scheme around v is consistent with $\mathcal{G}^-$. Note that such embeddings exists due to the previous condition.

Clearly, a wrung obstruction is not planar, because neither $\mathcal{G}^+$ nor $\mathcal{G}^-$ is an extension of $\mathcal{H}$. A *minimal wrung obstruction* is a wrung obstruction that does not contain a smaller wrung obstruction as a PEG-minor. A minimal wrung obstruction is not necessarily a minimal planarity obstruction—it may contain a smaller obstruction that is not wrung. However, it turns out that minimal wrung obstructions are close to being minimal planarity obstructions. The key idea in using wrung obstructions is that they are characterized by being subdivisions of 3-connected graphs, a property that is much easier to control than non-embeddability of PEGs.

The following proposition summarizes the key property of wrung obstructions. In particular, it shows that there only exist finitely many minimal wrung obstructions.

Proposition 7.2. *If $(G, H, \mathcal{H})$ is a minimal wrung obstruction, then every vertex of G also belongs to H and the graph H is connected.*

The proof of this proposition relies heavily on the notion of *contractible edge*, which is an edge in a 3-connected graph whose contraction leaves the graph 3-connected. This notion has been intensely studied [Kri00, Kri02], and we are able to use powerful structural theorems that guarantee that any 'sufficiently large' wrung obstruction must contain an edge that can be contracted to yield a smaller wrung obstruction.

Proof. Let $G^\star$ be the 3-connected graph whose subdivision is G. A *subdivision vertex* is a vertex of G of degree 2. A *subdivided edge* is path in G of length at least two whose every internal vertex is a subdividing vertex and whose endpoints are not subdividing vertices. Therefore, each edge of $G^\star$ either represents an edge of G or a subdivided edge of G.

The proof of the proposition is based on several claims.

Claim 7.1. Every subdividing vertex of G is a vertex of H. Every subdivided edge of G contains at most one vertex adjacent to u and at most one vertex adjacent to v. If H is disconnected then G has at most one subdivided edge, which (if it exists) connects u and v and is subdivided by a single vertex.

If G had a subdividing vertex x not belonging to H we could contract an edge of G incident to x to get a smaller PEG, which is still wrung. Two vertices adjacent to u in the same subdivided edge would imply the existence of a loop or a multiple edge in $G^\star$. If H is disconnected, then every vertex of H except for u and v has degree 1 in H. If a subdividing vertex adjacent to u were also adjacent to an H-neighbor of v, then the edge between them could be contracted. This proves the claim.

A fundamental tool in the analysis of minimal wrung obstructions is the concept of contractible edges. An edge e in a 3-connected graph F is *contractible* if $F.e$ is also 3-connected, where $F.e$ is the graph obtained from F by contracting e. Note that an edge in a 3-connected graph F is contractible if and only if $F - \{x, y\}$ is biconnected.

The following fact can be derived from a theorem by Kriesell [Kri00], see also [Kri02, Theorem 3].

Fact 1. If F is a 3-connected graph and w a vertex of F that is not incident with any contractible edge and such that $F - w$ is not a cycle, then w is adjacent to four vertices x_1, x_2, y_1, y_2, all having degree 3 in F, which induce two disjoint edges x_1y_1 and x_2y_2 of F, and both these edges are contractible.

We are now ready to show that every vertex of G also belongs to H. Suppose for a contradiction that G has a vertex w not belonging to H. By Claim 7.1, w is not a subdivision vertex, so w is also a vertex of $G^\star$. If w were incident to a contractible edge of

$G^\star$, we could contract this edge to obtain a smaller wrung obstruction. Hence, w is not incident to any contractible edge of $G^\star$. Fix now the four vertices from Fact 1, and let $e_1 = x_1y_1$ and $e_2 = x_2y_2$ be the two contractible edges. Necessarily all the four endpoints of e_1 and e_2 belong to H, otherwise we could contract one of them to get a smaller wrung obstruction. Moreover, the edges e_1 and e_2 cannot contain u or v, because their endpoints have degree three and are adjacent to the vertex w not belonging to H. Therefore, each endpoint of e_1 and e_2 is adjacent to either u or v in $G^\star$ (and also in G and in H).

Assume without loss of generality that x_1 is adjacent to u. Then y_1 cannot be adjacent to u, because then u and w would form a separating pair in $G^\star$, hence y_1 is adjacent to v. Analogously, we may assume that x_2 is adjacent to u and y_2 is adjacent to v. The graph H must be connected, otherwise we could contract e_1 or e_2. This means that H, together with e_1 and e_2 and the two edges wx_1 and wx_2 form a subdivision of K_4, and therefore they form a wrung obstruction properly contained in $(G, H, \mathcal{H})$. Therefore any vertex of G also belongs to H.

It remains to prove that H is connected. For this we need another concept for dealing with subdivisions of 3-connected graphs. Let F be a 3-connected graph and let $e = xy$ be an edge of F. The *cancellation* of e in F is the operation that proceeds in the three steps 1) Remove e from F, to obtain $F - e$ 2) If the vertex x has degree 2 in $F - e$, then replace the subdivided edge containing x by a single edge. Do the same for y as well. 3) Simplify the graph obtained from step 2 by removing multiple edges.

Let $F \ominus e$ denote the result of the cancellation of e in F. Note that $F \ominus e$ may contain vertices of degree 2 if they arise in step 3 of the above construction. An edge e is *cancellable* if $F \ominus e$ is 3-connected. It is called *properly cancellable* if it is cancellable, and moreover, the first two steps in the above definition produce a graph without multiple edges.

Claim 7.2. A cancellable edge e in a 3-connected graph F is either properly cancellable or contractible.

Suppose that $e = xy$ is cancellable, but not properly cancellable. We show that it is contractible. Since e is not properly cancellable, one of its endpoints, say x has degree 3 in F and its two neighbors x' and x'' besides y are connected by an edge. We show that between any pair of vertices a and b of $F - \{x, y\}$ there are two vertex-disjoint paths. In F there exist three vertex-disjoint $a - b$-paths P_1, P_2 and P_3. If two of them avoid x and y then they are also present in $F - \{x, y\}$. Therefore, we may assume that P_1 contains x and P_2 contains y. Then P_1 contains the subpath $x'xx''$ which can be replaced by the single edge $x'x''$. Again at most one of the paths contains vertices of $\{x, y\}$ and therefore we again find two vertex-disjoint $a - b$-paths in $F - \{x, y\}$. This shows that $F - \{x, y\}$ is biconnected and therefore $e = xy$ is contractible. This concludes the proof of the claim.

Moreover, we need the following result by Holton et al. [HJSW90].

Fact 2. If F is a 3-connected graph with at least five vertices, then every triangle in F has at least two cancellable edges.

We are now ready to show that H is connected. Suppose for a contradiction that H is disconnected, and let H_u and H_v be its two components containing u and v. Let x_1, x_2 and x_3 be the three neighbors of u in H, and y_1, y_2 and y_3 the three neighbors of v. Recall from Claim 7.1 that G has at most one subdividing vertex, and that the possible subdivided edge connects u and v.

Since $G^\star$ is 3-connected, it has three disjoint edges e_1, e_2 and e_3 connecting a vertex of H_u to a vertex of H_v. At least one of them avoids both u and v. Assume without loss of generality that $e_1 = x_1y_2$ is such an edge. If e_1 were a contractible edge of $G^\star$, we

would get a smaller wrung obstruction. Therefore the graph $G^\star - \{x_1, y_1\}$ has a cutvertex w. Note that w is either u or v. Otherwise, $H_u - \{x_1, y_1, w\}$ would be connected, and also the graph $H_v - \{x_1, y_1, w\}$ and at least one of the edges e_2, e_3 avoids w, showing that $G^\star - \{x_1, y_1, w\}$ is connected.

So, without loss of generality, $G^\star$ has a separating triplet $\{x_1, y_1, u\}$. Since at least one of the two edges e_2, e_3 avoids this triplet, we see that one of the components of $G^\star - \{x_1, y_1, u\}$ consists of a single vertex $x' \in \{x_2, x_3\}$. Since each vertex in a minimal separator must be adjacent to each of the components separated by the separator, $G^\star$ contains the two edges $x'x_1$ and $x'y_1$. Consequently, x', x_1 and y_1 induce a triangle in $G^\star$ (and in G), and by Fact 2, at least one of the two edges x_1y_1 and $x'y_1$ is cancellable, and by Claim 7.2, at least one of the two edges is contractible or properly cancellable, contradicting the minimality of $(G, H, \mathcal{H})$. This completes the proof of Proposition 7.2. □

Proposition 7.2 implies that a minimal wrung obstruction has at most seven vertices. Therefore, to show that each wrung obstruction contains one of the minimal obstructions from Figure 7.1 is a matter of a finite (even if a bit tedious) case analysis. We remark that a minimal wrung obstruction may contain any of the exceptional obstructions of Figure 7.1, except obstructions 18–22, obstruction 3, K_5, and $K_{3,3}$. A minimal wrung obstruction does not contain any k-fold alternating chain for $k \geq 4$. As the analysis requires some more techniques, we defer the proof to Lemma 7.8.

Let us show how the concept of wrung obstruction can be applied in the analysis of R-skeletons. Consider again the skeleton $\mathfrak{R}$, with two distinct vertices u and v, each of them violating edge-compatibility of one of the two embeddings of $\mathfrak{R}$. This means that u is incident to three H-edges e_1, e_2, e_3 projecting into distinct $\mathfrak{R}$-edges e'_1, e'_2, e'_3, such that the cyclic order of e_i's in $\mathcal{H}$ coincides with the cyclic order of e'_i's in $\mathfrak{R}^-$, and similarly v is adjacent to H-edges f_1, f_2, f_3 projecting into $\mathfrak{R}$-edges f'_1, f'_2, f'_3, whose order in $\mathfrak{R}^+$ agrees with $\mathcal{H}$.

It all the e'_i and f'_i for $i = 1, 2, 3$ are distinct, then it is fairly easy to see that G must contain a wrung obstruction, obtained simply by replacing each edge of $\mathfrak{R}$ with a path of G, chosen in such a way that all the six edges e_i and f_i belong to these paths. Such a choice is always possible and yields a wrung obstruction. In particular, this is always the case if u and v are not adjacent in $\mathfrak{R}$.

If, however, u and v are connected by an $\mathfrak{R}$-edge g', and if, moreover, we have $e'_i = g' = f'_j$ for some i and j, the situation is more complicated, because there does not have to be a path in G that contains both edges e_i and f_j and projects into g'. In such a situation, we do not necessarily obtain a wrung obstruction. This situation is handled separately in Lemma 7.7. Altogether we prove the following proposition.

Proposition 7.3. *Let $(G, H, \mathcal{H})$ be a biconnected obstruction-free* Peg*, and let $\mathfrak{R}$ be the skeleton of an R-node of the SPQR-tree of G. Then $\mathfrak{R}$ has an edge-compatible embedding.*

This essentially concludes our treatment of the case when $\mathfrak{R}$ has no edge-compatible embedding. We already know that if $\mathfrak{R}$ does not have an edge-compatible embedding, then it either contains obstruction 2, or two distinct vertices u and v requiring different embeddings of $\mathfrak{R}$. In this case $(G, H, \mathcal{H})$ either contains a wrung obstruction, or it does not, and u and v are connected by an edge.

In the remainder of this subsection, we prove that in either case $(G, H, \mathcal{H})$ contains one of the obstructions from Figure 7.1. We first show that if $\mathfrak{R}$ does not contain a wrung obstruction, then it contains one of the obstructions 4,5 and 6; see Lemma 7.7. Finally, we also present a detailed analysis showing that any minimal wrung obstruction (for which

we already know that there are only finitely many) contains one of the obstructions from Figure 7.1; see Lemma 7.8.

The following technical lemma is a useful tool, which we employ in both proofs. To state the lemma, we use the following notation: let $x_1, x_2, \ldots, x_k$ be (not necessarily distinct) vertices of a graph F. We say that a path P in F is *a path of the form* $x_1 \to x_2 \to \cdots \to x_k$, if P is a path obtained by concatenating a sequence of internally disjoint simple paths $P_1, P_2, \ldots, P_{k-1}$, where P_i is a path connecting x_i to x_{i+1} (note that if $x_i = x_{i+1}$, then P_i consists of a single vertex).

Lemma 7.6. *Let $\mathfrak{R}$ be a 3-connected graph with a fixed planar embedding $\mathfrak{R}^+$. Let u and v be two vertices of $\mathfrak{R}$ connected by an edge B. Let u_1 and u_2 be two distinct neighbors of u, both different from v, such that (v, u_1, u_2) appears counter-clockwise in the rotation scheme of u. Similarly, let v_1 and v_2 be two neighbors of v such that (u, v_1, v_2) appears clockwise around v. (Note that we allow some of the u_i to coincide with some v_j.) Then at least one of the following possibilities holds:*

1. *The graph $\mathfrak{R}$ contains a path of the form $v \to u_1 \to u_2 \to v_1 \to v_2 \to u$.*
2. *The graph $\mathfrak{R}$ contains a path of the form $u \to v_1 \to v_2 \to u_1 \to u_2 \to v$. (This is symmetric to the previous case.)*
3. *The graph $\mathfrak{R}$ has a vertex w different from u_2 and three paths of the forms $w \to u_2 \to v_2$, $w \to u_1$ and $w \to v_1$, respectively. These paths only intersect in w, and none of them contains u or v.*
4. *The graph $\mathfrak{R}$ has a vertex w different from v_2 and three paths of the forms $w \to v_2 \to u_2$, $w \to v_1$ and $w \to u_1$, respectively. These paths only intersect in w, and none of them contains u or v. (This is again symmetric to the previous case.)*

Proof. Let C_u be the horizon of u and C_v the horizon of v. Orient C_u counterclockwise and split it into three internally disjoint oriented paths $v \to u_1$, $u_1 \to u_2$ and $u_2 \to v$, denoted by C_u^1, C_u^2, and C_u^3 respectively. Similarly, orient C_v clockwise, and split it into $C_v^1 = u \to v_1$, $C_v^2 = v_1 \to v_2$, and $C_v^3 = v_2 \to u$.

Let F_1 and F_2 be the two faces of $\mathfrak{R}$ incident with the edge uv, with F_1 to the left of the directed edge $\vec{uv}$. Note that each vertex on the boundary of F_1 except u and v appears both in C_u^1 and in C_v^1. Similarly, the vertices of F_2 (other than u and v) appear in C_u^3 and C_v^3. There may be other vertices shared between C_u and C_v and we have no control about their position. However, at least their relative order must be consistent, as shown by the following claim.

Claim. Suppose that x and y are two vertices from $C_u \cap C_v$, at most one of them incident with F_1 and at most one of them incident with F_2. Then (v, x, y) are counter-clockwise on C_u if and only if (u, x, y) are clockwise on C_v; see Figure 7.4.

Proof. Draw two curves γ_x and γ_y connecting u to x and to y, respectively. Draw similarly two curves δ_x and δ_y from v to x and to y. The endpoints of each of the curves appear in a common face of $\mathfrak{R}^+$, so each curve can be drawn without intersecting any edge of $\mathfrak{R}$. Also, the assumptions of the claim guarantee that at most two of these curves can be in a common face of $\mathfrak{R}^+$, and this happens only if they share an endpoint, so the curves can be drawn internally disjoint. Consider the closed curve formed by γ_x, δ_x, and the edge uv, oriented in the direction $u \to x \to v \to u$. Suppose, w.l.o.g., that y is to the left of this closed curve. Then γ_y is also to the left of it, and (uv, γ_x, γ_y) appear in counter-clockwise order around u, so (v, x, y) are counter-clockwise on C_u. By analogous reasoning, (u, x, y) are clockwise around v. The claim is proved. □

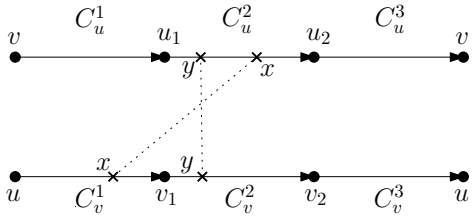


Figure 7.4.: The two directed horizontal lines represent C_u and C_v. A vertex x appearing on both C_u and C_v is represented by a dotted line connecting its position on C_u with its position on C_v. Here is an example of a situation forbidden by the Claim, where two shared vertices x and y appear in different order on the directed cycles C_u and C_v.

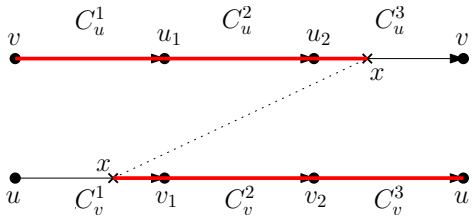


Figure 7.5.: Case A of the proof of Lemma 7.6. The red line represents the constructed walk.

We now consider several cases depending on whether various parts of C_u share vertices with parts of C_v.

Case A. C_u^3 shares a vertex x with C_v^1. Consider a walk starting in v, following C_u counter-clockwise through u_1 and u_2 until x, then following C_v clockwise from x through v_1 and v_2 till u; see Figure 7.5. The above Claim guarantees that this walk is actually a path (note that x cannot belong to either F_1 or F_2). This path corresponds to the first case in the statement of the lemma. Similarly, if $C_u^1 \cap C_v^3$ is nonempty, a symmetric argument yields the second case of the lemma.

Assume for the rest of the proof that $C_u^3 \cap C_v^1 = \emptyset = C_u^1 \cap C_v^3$.

Case B. No internal vertex of C_u^2 belongs to $C_v^1 \cup C_v^3$. Define a walk in $\mathfrak{R}$ by starting in v_1, following C_v counter-clockwise until we reach the first vertex (call it x) that belongs to C_u, then following C_u counter-clockwise through u_1 and u_2, until we reach the first vertex y from $C_u^3 \cap C_v$, then following C_v from y towards v_2 while avoiding v_1 and u. Note that the vertices x and y must exist, because F_1 and F_2 each have at least one vertex from $C_u \cap C_v$. Note also that $x \in C_u^1$ and $y \notin C_v^1$, otherwise we are in Case A.

The walk defined above is again a path, it avoids u and v, and by putting $w := u_1$, we are in the situation of the third case of the lemma. Symmetrically, if no internal vertex of C_v^2 belongs to $C_u^1 \cup C_u^3$, we obtain the fourth case of the lemma.

Suppose that none of the previous cases (and their symmetric variants) occurs. What is

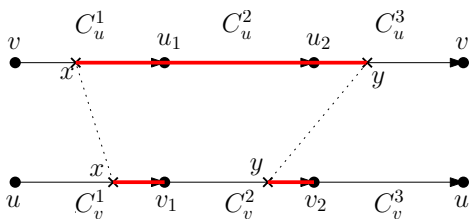


Figure 7.6.: Case B of the proof of Lemma 7.6. Note that the vertex y may also belong to C_v^3.

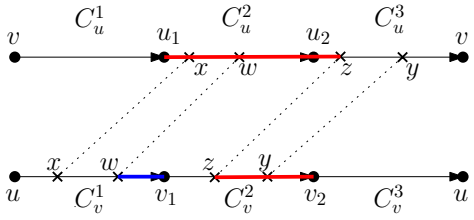


Figure 7.7.: Case C of the proof of Lemma 7.6. The red line is W_1, the blue line is W_2.

left is the following situation.

Case C. C_u^2 has an internal vertex x belonging to $C_v^1 \cup C_v^3$, and C_v^2 has an internal vertex y belonging to $C_u^1 \cup C_u^3$. We cannot have $x \in C_v^1$ and $y \in C_u^1$ as that would contradict the Claim above. So assume that $x \in C_v^1$ and $y \in C_u^3$, the other case being symmetric. Consider a walk W_1 from u_1 along C_u counter-clockwise through u_2, and let z be the first vertex on C_u after u_2 that belongs to C_v. We must have $z \in C_v^2$, otherwise z and y violate the Claim. Continue W_1 from z till v_2 clockwise along C_v. Start another walk W_2 from v_1 counterclockwise along C_v and let w be the first vertex of C_u encountered. Necessarily $w \in C_u^2$, otherwise x and w violate the Claim. Therefore, $w \in W_1 \cap W_2$, and w is the only such vertex. This results in case 3 from the lemma. □

A straightforward case analysis based on Lemma 7.6 shows that if $(G, H, \mathcal{H})$ has no wrung obstruction, then it must contain obstruction 4, 5, or 6.

Lemma 7.7. *Let $(G, H, \mathcal{H})$ be a* PEG *and let $\mathfrak{R}$ be the skeleton of an R-node of G such that $\mathfrak{R}$ has two distinct vertices u and v, each violating edge-compatibility of one of the embeddings of $\mathfrak{R}$. If G does not have a wrung obstruction where u and v have H-degree 3, then $(G, H, \mathcal{H})$ contains obstruction 4, 5, or 6.*

Proof. As we have seen, u must be incident to three H-edges $e_1 = ux_1, e_2 = ux_2, e_3 = ux_3$ projecting to distinct $\mathfrak{R}$-edges $e_1' = uu_1, e_2 = uu_2', e_3 = uu_3'$, such that the cyclic order of the e_i's in $\mathcal{H}$ coincides with the cyclic order of e_1', e_2', e_3' in $\mathfrak{R}^+$. Similarly, v is incident to three H-edges $f_1 = vy_1, f_2 = vy_2, f_3 = vy_3$ projecting to distinct $\mathfrak{R}$-edges $f_1' = vv_1, f_2' = vv_2, f_3' = vv_3$ whose order in $\mathfrak{R}^-$ agrees with $\mathcal{H}$. As we have seen, we obtain a wrung obstruction if all the edges e_1', e_2', e_3' and f_1', f_2', f_3' are distinct. Hence, one edge e_i' must coincide with an edges f_j'. After possibly renaming the edges, we can assume $e_3' = f_3'$, and hence $u_3 = v$ and $v_3 = u$. We consider the embedding $\mathfrak{R}^+$ of $\mathfrak{R}$ where without loss of generality the counterclockwise around u is (e_1', e_2', e_3').

We now distinguish several cases, based on whether some of the u_i and v_j for $1 \leq i, j \leq 2$ coincide. The general structure of the analysis is as follows.

I) The vertices u_1, u_2, v_1, and v_2 are not distinct.

After possibly swapping u and v and/or mirroring the embedding, we may assume that u_1 coincides with one of the other vertices.

a) $u_1 = v_1$

1) Additionally, we have $u_2 = v_2$.

We show that if H contains the edges uu_1, uu_2, vv_1 and v, v_2, then we find obstruction 4. Otherwise we get obstruction 4.

2) If the vertices u_2 and v_2 are distinct we obtain obstruction 5.

b) If u_1 coincides with v_2, then u_2 and v_1 must be distinct and we obtain obstruction 6.

II) If the vertices u_1, u_2, v_1 and v_2 are distinct, we obtain obstruction 6.

We now present the details of this analysis. As a first step, we unprescribe all H-edges, except for the e_i and f_i, for $i = 1, 2, 3$. Let P be a path from x_3 to v in G that projects to uv in $\mathfrak{R}$. Analogously, let Q be a path from v_3 to u in G, projecting to uv. The paths P and Q are disjoint, otherwise there would be a path connecting x to y, while avoiding u and v, which would imply the existence of a wrung obstruction. In the following we will always contract P and Q to single edges, hence all obstructions we consider will contain the edges x_3v and y_3u. Let A be a path of length at least 2 in $\mathfrak{R}$ connecting two vertices a and b. A *representative path* of A is a path A' in G such that all edges and vertices of A' belong to an edge of A. Additionally, if the first edge of A contains an H-edge incident to a, we require that A' starts with an H-edge. Analogously, if the last edge of A contains an H-edge incident to b, we require that A' ends with an H-edge. The properties of the SPQR-tree decomposition imply that for any path A in $\mathfrak{R}$ a representative path exists. An *internal edge* of a path A is an edge of A that is neither the first nor the last edge of A.

Case I: The vertices u_1, u_2, v_1 and v_2 are not distinct. As stated above, we can assume without loss of generality that u_1 coincides with one of the other vertices. Since u_1 and u_2 are endpoints of distinct edges sharing the endpoint u, they cannot coincide, and hence we either have $u_1 = v_1$ or $u_1 = v_2$.

Case Ia: We now assume that $u_1 = v_1$. The next case distinction is on whether u_2 and v_2 are distinct or not.

Case Ia1: We have $u_1 = v_1$ and $u_2 = v_2$. Since $\mathfrak{R}$ is 3-connected, it contains a path T from u_1 to u_2 that avoids both u and v. Let T' be a representative path of T.

Assume now that H contains the edges uu_1, uu_2, vu_1, vu_2, that is they form a cycle of length 4. We denote this cycle by C. We remove from G all vertices that are not part of P, Q, T' or C. The incompatibility of the rotation schemes of u and v imply that x_3 and y_3 are embedded on distinct sides of C in $\mathcal{H}$. Hence, after contracting P, Q and T to a single G-edge, each, we obtain obstruction 4.

Now assume that not all the above edges are part of H. Without loss of generality we can assume that the edge uu_2 is not in H. Let A be a representative path of uu_1v, and hence starting with e_1 and ending with f_1. Analogously, let B be a representative path of uu_2v, starting with e_2 and ending with f_2. We now remove from G all vertices that do not belong to any of the paths P, Q, T', A and B. We then contract T' to a single edge, using only simple G-edge contractions.

Since the edge uu_2 is not in H, we know that B has at least three edges, we contract its internal edges to the single edge x_2y_2. The path A starts with the edge ux_1 and ends with y_1v. If it has length at least 3, we first contract its internal edges to the single edge x_1y_1, using simple G-edge contractions. Since in this case, x_1 and y_1 belong to distinct components of H, and both have degree 1, we may additionally contract the edge x_1y_1 to a single vertex using the complicated G-edge contraction. Hence, after these steps the path A consists of the two H-edges ux_1 and x_1v. Contracting P and Q to single edges yields obstruction 5.

Case Ia2: We have $u_1 = v_1$, but u_2 and v_2 are distinct. Let again A be a representative path of uu_1v. Since $\mathfrak{R}$ is 3-connected and planar, it is not hard to see that it must contain a path T from u_2 to v_2 that avoids u, v and u_1. Moreover, since $\mathfrak{R}$ is 3-connected there exists a path S from a vertex $t \in T$ to u_1 that avoids u and v and is internally disjoint from T. Since $u_2 \neq v_2$ it holds that $t \neq u_2$ or $t \neq v_2$ (or both). We assume that $t \neq u_2$,

the other case is symmetric. Let T' be a representative path of the path obtained by concatenating uu_2, T and v_2v, and let S' be a representative path of S.

To obtain an obstruction, we first remove all vertices of G that do not belong to either of A, P, Q, S' and T'. As above, we contract A to the path ux_1v, which is allowed since x_1 and y_1 are in distinct connected components of H, if A has any internal edges. Further, we contract the subpath of T' from t to v_2 to the single vertex v_2 (this is allowed since $t \notin H$, unless $t = v_2$) and then we contract the subpath of T' from u_2 to $t = v_2$ to the single edge u_2v_2 (recall that $u_2 \neq t$). Finally, we contract the path S' to the single edge u_1v_2 (recall that $t = v_2$ after the previous contractions). Contracting the paths P and Q to a single edge, each, yields obstruction 5.

Case Ib: The two vertices u_1 and v_2 coincide. The rotation schemes of u and v imply that u_2 and v_1 are embedded on different sides of the cycle uu_1v in $\mathfrak{R}^+$, and can therefore not coincide. Since $\mathfrak{R}$ is 3-connected it contains disjoint paths from u_2 to each of u_1, v, avoiding u. Analogously, $\mathfrak{R}$ contains disjoint paths from v_1 to u, v_2, avoiding v and since u_2 and v_1 are on different sides of the cycle uu_1v, these paths are all disjoint. We define the path A as before and contract it to the two prescribed edges ux_1v. Further, we take a representative of each of the above paths and contract it to a single G-edge. Together with contracting P and Q to single edges, this yields obstruction 6.

Case II: Assume that the vertices u_1, v_1, u_2 and v_2 are distinct. We now apply Lemma 7.6 (the naming of the vertices is chosen as in the lemma).

If case 1 of the lemma applies, we obtain a simple path T in $\mathfrak{R}$ visiting the vertices v, u_1, u_2, v_1, v_2 and u in this order. Let A be a representative path composed of uu_2, the subpath of T from u_2 to v_1 and v_1u. Let T be a representative path of the path composed of uu_1 and the subpath of T from u_1 to v. Symmetrically, let C be a representative path of the path composed of vv_2 and the subpath of T from v_2 to u. Further, let C and D be representative paths of the subpaths of T from u_1 to v_1 and from u_2 to v_2, respectively (if one of the subpaths of T consists of only one edge, we simply take any G-path between the endpoints that is contained in the corresponding subgraph of G). We now remove from G all vertices that are not part of one of these paths or the paths P and Q. As above the vertices x_2 and y_1 belong to distinct components of H, therefore we can contract the internal edges of A to the single vertex $u_2 = x_2 = y_1 = v_1$, so that A becomes ux_1v. Next, we contract the internal edges of B to a single edge, so that B becomes ux_1v, and in particular we get $u_1 = x_1$. Analogously, we contract C to vy_2u such that $v_2 = y_2$. Finally, we contract C and D to the single edges x_1x_2 and y_1y_2, respectively, and contract P and Q to single edges. This yields obstruction 6. Case 2 of Lemma 7.6 is completely symmetric.

If case 3 of the lemma applies, there exists a vertex $w \neq u_2$ of $\mathfrak{R}$ that has internally disjoint paths A, B, and C from w to u_1, from w to v_1 and from w via u_2 to v_2, respectively, and all of them avoid u and v. Note that Lemma 7.6 guarantees $u_2 \neq w \neq v_2$.

Let A' be a representative of the path composed of uu_1 and A, and let B' be a representative path of the path composed of vv_1 and B. Let C' be a representative path the path obtained by concatenation of uu_2, the subpath of C from u_2 to v_2 and v_2v, and let D' be a representative of the subpath of C from w to u_2. Again, we remove all vertices of $V(G) \setminus V(H)$ that are not part of any of our paths. We contract the internal edges of C' to a single vertex, so that it becomes the path ux_2y_2v, and in particular we get $x_2 = u_2 = v_2 = y_2$. We contract A' to the path ux_1w (we get $x_1 = u_1$), and analogously B' to vy_1w (with $u_1 = y_1$). Note that we may have $w = x_1$ or $w = y_1$, but not both. Finally, we contract C' to the edge wx_2 and P and Q to single edges. If $w = u_1$ of $w = u_2$ this directly yields obstruction 5, otherwise we obtain obstruction 5

by contracting u_1w to a single vertex, which is allowed since w is not in H.

Since case 4 of Lemma 7.6 is completely symmetric to case 3 this finishes the case analysis and thus concludes the proof. □

The following lemma shows that any minimal wrung obstruction contains one of the obstructions in Figure 7.1. Although the analysis is straight-forward, there exist many different cases. For the sake of completeness, we provide a detailed proof.

Lemma 7.8. *A minimal wrung obstruction contains one of the obstructions 1,2, or 4–17.*

Proof. Let $(G, H, \mathcal{H})$ be a minimal wrung obstruction, and let $\mathfrak{R}^+$ be a planar embedding of G. By the definition of a wrung obstruction, the graph H contains a vertex u with three adjacent vertices x_1, x_2, x_3 occurring in this counterclockwise order around u both in $\mathfrak{R}^+$ and in $\mathcal{H}$. Similarly, there exists a second vertex v distinct from u that has three adjacent vertices y_1, y_2, y_3 that occur in this clockwise order in $\mathfrak{R}^+$ but in the reverse clockwise order in $\mathcal{H}$. All other edges and vertices belong to G. Moreover, G is a subdivision of a 3-connected graph.

By Proposition 7.2 the graph H is connected, and all vertices of G also belong to H. The proof itself is split into two parts. In the first part, we assume that H contains the edge uv, whereas in the second we assume that this edge does not belong to H. Each of these two parts consists of a case analysis for which we will first give a sketch and then a detailed description of each individual case.

In the first part, we assume that the edge uv is in H. Then, since u and v have degree 3 in H, we may, after possibly renaming the vertices, assume that $x_3 = v$ and $y_3 = u$. We now distinguish several cases based on which of the other vertices coincide. The outline of the case analysis is as follows.

I) Some of the vertices x_1, x_2, y_1 and y_2 coincide.

After renaming and possibly exchanging the embedding $\mathfrak{R}^+$ with $\mathfrak{R}^-$, we may assume that x_1 coincides with one of the other vertices. Since x_1 and x_2 are distinct neighbors of u they cannot coincide and therefore we either have $x_1 = y_1$ or $x_1 = y_2$. We distinguish these cases.

a) If $x_1 = y_1$, we show that $(G, H, \mathcal{H})$ contains obstruction 1.

b) If $x_1 = y_2$, we show that $(G, H, \mathcal{H})$ contains obstruction 17.

II) The vertices x_1, x_2, y_1 and y_2 are distinct.

In this case, we make a case distinction on whether one of these vertices is a subdivision vertex, that is, it has degree 2 in G. Again, we may assume that if any vertex is a subdivision vertex, then u_1 is one.

a) One of the above vertices is a subdivision vertex, without loss of generality x_1.

Note that if x_1 is a subdivision vertex then its two incident edges may not be part of a triangle, as this would create parallel edges when replacing the subdivided edge with a single one. Hence, x_1 must be connected either to y_1 or to y_2. We claim that y_1 cannot be a subdivision vertex, too. Assume this is the case. Then G cannot contain the edge x_1y_1, as this would be a subdivision of the edge uv, which already is in the graph. Therefore it must contain x_1y_2. However, since y_1 is also assumed to be a subdivision vertex, it must also contain one of the edges x_1y_1 or x_2y_1. The former is not allowed since x_1 is a subdivision vertex and the latter would violate the planarity of $\mathfrak{R}^+$. Therefore this case can be excluded.

1) If the vertices x_1 and x_2 are subdivision vertices, we find obstruction 2.
2) If the vertices x_1 and y_2 are subdivision vertices, we find obstruction 2.
3) If only x_1 is a subdivision vertex, we obtain either obstruction 2 or 10.

b) None of the above vertices is a subdivision vertex.

In this case we apply Lemma 7.6, and obtain either obstruction 2 or 7.

We now present a detailed analysis of these cases.

Case I: In this case, we assume that some of the vertices x_1, x_2, y_1 and y_2 coincide. As argued above, we can assume without loss of generality that x_1 coincides with either y_1 or with y_2.

Case Ia: We have $x_1 = y_1$. Note that x_2 and y_2 must be distinct since H has at least one vertex of degree 1 and that further the rotation schemes of u and v imply that x_2 and y_2 lie on the same side of the H-cycle ux_1v in $\mathfrak{R}^+$, and thus on different side in $\mathcal{H}$. We now argue that G must contain the edge x_2y_2. If x_2 was a subdivision vertex, then it could not be connected to either of u, x_1 and v as this would form a triangle. Hence in this case x_2 must be connected to y_2. A symmetric argument holds if y_2 is a subdivision vertex. Therefore, if x_2y_2 is not in the graph, both vertices must have degree at least 3. However, since x_2 and y_2 are embedded on the same side of the triangle ux_1v in the planar embedding $\mathfrak{R}^+$, it cannot contain all the edges $x_2u, x_2x_1, x_2v, y_2u, y_2x_1$ and y_2v (otherwise inserting a new vertex w on the other side of the triangle ux_1v and connecting it to all vertices of the triangle would yield a planar drawing of $K_{3,3}$). Hence, in order for x_2 and y_2 to have degree 3, the edge x_2y_2 must belong to G. Recall that since x_2 and y_2 are one the same side of the triangle ux_1v in $\mathfrak{R}^+$, they are on different sides in $\mathcal{H}$, and hence the triangle ux_1v together with the edge x_2y_2 form obstruction 1.

Case Ib: We have $x_1 = y_2$. Let T denote the triangle ux_1v. The rotation schemes of u and v imply that y_2 and y_2 are embedded on different sides of T in $\mathfrak{R}^+$. This implies that none of them can be a subdivision vertex, as their incident edges would be part of a triangle. Hence G must contain the edges x_2x_1, x_2v, y_2u and y_1y_2. The vertices x_2 and y_2 are embedded on different side of T in $\mathcal{H}$, and hence after unprescribing the edges x_2u and y_2v, the triangle T together with the remaining edges forms obstruction 17. This concludes the cases where x_1 coincides with one of the other vertices.

Case II: The vertices x_1, x_2, y_1 and y_2 are all distinct. We distinguish cases based on whether any of them is a subdivision vertex.

Case IIa: One of x_1, x_2, y_1 and y_2 is a subdivision vertex. We may assume without loss of generality that x_1 is one. As argued above, the graph G must contain one of the edges x_1y_1 or x_1y_2.

Case IIa1: Both x_1 and x_2 are subdivision vertices. The vertices x_1 and x_2 must be connected to one of the vertices y_1 or y_2, each. If they were connected to the same vertex, they would form different subdivisions of the same edge. Hence, they must be connected to distinct vertices. The edge x_1y_2 would not allow for such a connection in a planar way, therefore G must contain x_1y_1 and x_2y_2. After contracting uv, this yields obstruction 2.

Case IIa2: Both x_1 and y_2 are subdivision vertices. The graph G cannot contain the edge x_1y_2 as this would be a subdivision of an edge parallel to uv, which is already in G. Therefore G contains the edge x_1y_1. Since y_2, as a subdivision vertex, must be adjacent to x_1 or x_2, and the former case is excluded as x_1 is a subdivision vertex, G contains the edge x_2y_2. Again, contracting uv yields obstruction 2.

Case IIa3: Only x_1 is a subdivision vertex. We distinguish whether x_1y_1 or x_1y_2 is in G.

Assume that x_1y_1 is in G. The rotation schemes of u and v imply that x_2 and y_2 are on the same side of the cycle formed by ux_1y_1vu in $\mathfrak{R}^+$. As in case Ia) we can argue that x_2y_2 must belong to G, too. As in the previous case this yields obstruction 2.

Now assume that $x_1y_2 \in E(G)$. By the rotation schemes of u and v the vertices x_2 and y_1 lie on distinct side of the cycle formed by ux_1y_2v. Further, they must have degree 3 and therefore the edges x_2v, x_2y_2, y_1u, and y_1y_2 must be in G. Altogether this forms obstruction 10.

This finishes the cases where the vertices x_1, x_2, y_1 and y_2 are distinct and one of them is a subdivision vertex.

Case IIb: The vertices x_1, x_2, y_1 and y_2 are distinct and each of them has degree at least 3. In particular G does not have subdivision vertices, and thus is 3-connected. We can therefore apply Lemma 7.6 (the naming of the vertices is chosen as in the lemma). We distinguish, which of the cases of the lemma applies.

If case 1 (or symmetrically case 2) of the lemma applies, G contains the path $v \to x_1 \to x_2 \to y_1 \to y_2 \to u$ forming obstruction 7.

If case 3 (or symmetrically case 4) of the lemma applies, we obtain a vertex $w \neq x_2$ and paths $w \to x_2 \to y_2$, $w \to x_1$, and $w \to y_1$. Since G has only six vertices, w and all edges avoid u and v, it follows that w must coincide with x_1 or y_1. In both cases G contains the two edges x_1y_1 and x_2y_2, which yields obstruction 2. This concludes the analysis of minimal wrung obstructions where u and v are adjacent.

In the second part of the proof we consider minimal wrung obstructions where u and v are not adjacent in H. Since H is connected, one of the x_i must coincide with one of the y_j and after renumbering them, we may assume that $x_3 = y_3$. To obtain a more symmetric notation where this vertex is not notationally biased towards u or v, we denote it by w. We now make a case distinction, based on which of the vertices x_1, x_2, y_1 and y_2 are distinct and which ones are subdivision vertices. The overall case analysis works as follows.

I) Some of the vertices x_1, x_2, y_1, y_2 coincide.

   Similar to before, by symmetry we can assume that x_1 coincides with one of y_1 or y_2.

   a) If $x_1 = y_1$, then $(G, H, \mathcal{H})$ contains obstruction 1 or 4.

   b) If $x_1 = y_2$, then $(G, H, \mathcal{H})$ contains obstruction 16.

II) The vertices x_1, x_2, y_1, y_2 are all distinct and the wrung obstruction has a vertex of degree 2. Note that w may not be a subdivision vertex, otherwise we could contract its incident edges to the edge uv to obtain a smaller wrung obstruction. Therefore one of the vertices x_1, x_2, y_1 and y_2 must be a subdivision vertex. By symmetry we may again assume that x_1 is a subdivision vertex. We consider several subcases.

   a) If the vertex y_1 is also subdividing and G contains the edge x_1y_1, then $(G, H, \mathcal{H})$ contains obstruction 2.

   b) If the vertex y_2 is also subdividing and G contains the edge x_1y_2, then $(G, H, \mathcal{H})$ contains obstruction 12.

      By symmetry these two cases cover all situations in which two subdivision vertices are connected by an edge.

   c) The graph G contains the edge x_1v.

      Here we distinguish several subcases, depending on which other vertices are subdividing.

1) If x_2 is also subdividing and adjacent to y_1, then $(G, H, \mathcal{H})$ contains obstruction 13.
2) If x_2 is also subdividing and adjacent to y_2, $(G, H, \mathcal{H})$ contains obstruction 5.
3) If y_1 is subdividing and adjacent to x_2, $(G, H, \mathcal{H})$ contains obstruction 13.
4) If y_2 is subdividing and adjacent to x_2, $(G, H, \mathcal{H})$ contains obstruction 5.

   Note that x_2 may not be subdividing and adjacent to v, as it would be parallel to the edge subdivided by x_1. For the same reason, y_1 (respectively y_2) may not be subdividing and adjacent to u. Hence this covers all the subcases where another vertex except for x_1 is a subdivision vertex.
5) If no other vertex is subdividing, $(G, H, \mathcal{H})$ contains one of the obstructions 5,9,13 and 14.

d) The vertex x_1 is subdividing and adjacent to y_1, but y_1 is not subdividing. Again, we consider subcases, based on other subdividing vertices and their adjacencies.
1) If x_2 is subdividing and adjacent to y_2, we find obstruction 2.
2) If y_2 is subdividing and adjacent to y_2, we find obstruction 2.

   Note that this covers all the cases where any other vertex is subdividing. If x_2 was subdividing and adjacent to v, we could exchange x_2 with x_1 to obtain an instance of case IIb. Analogously for y_2 subdividing and adjacent to u. Further, x_2 cannot be subdividing and adjacent to y_1 as this would create parallel subdivided edges ux_1y_1 and ux_2y_1. Therefore this covers all subcases where another vertex except x_1 is a subdivision vertex.
3) If no vertex besides x_1 is subdividing, we find obstruction 5 or 15.

e) If x_1 is subdividing and adjacent to y_2, but y_2 is not subdividing, we find obstruction 9 or 13.

   This does not need specific subcases, as no other vertex can be subdividing. If x_2 was subdividing, it would be adjacent to v and by mirroring the embedding and exchanging x_1 with x_2 and y_1 with y_2, we would arrive in case IIc. Analogously for y_2, which would have to be adjacent to u. Hence we can assume that x_1 is the only subdivision vertex.

III) The vertices x_1, x_2, y_1, y_2 are all distinct and all vertices in the wrung obstruction have degree at least 3.

a) G contains the edge uv.

   We distinguish cases, based on the embedding of the edge uv, by considering the relative positions of the cycle $C = uwvu$ and the vertices x_1, x_2, y_1 and y_2.
1) If all these vertices are on the same side of C, we find obstruction 2,7,8 or 15.
2) If one of these vertices is on one side and the others are on the other side, we obtain obstruction 5,9 or 13. In this case, we may assume without loss of generality that C separates x_1 from the other vertices.
3) If the cycle C separates x_1 and x_2 from the other vertices, we find obstruction 11.
4) If the cycle C separates x_1 and y_1 from the other vertices, we find obstruction 2.

   All other cases are symmetric to one of these.

b) G does not contain the edge uv.

Here, we use the fact that G is 3-connected, and thus contains three vertex-disjoint paths p_1, p_2 and p_3 from $\{x_1, u, x_2\}$ to $\{y_1, v, y_2\}$. We distinguish cases, based on which vertex is connected to which.

1) The path p_1 connects x_1 to y_1, p_2 connects u to v; we obtain obstruction 2.
2) The path p_1 connects x_1 to y_1, p_2 connects u to y_2; we obtain obstruction 2,5 or 9.
3) The path p_1 connects x_1 to v, p_2 connects x_2 to y_1; we obtain obstruction 2,9,11,12 or 13.

   This covers all cases where the paths connect x_1 to y_1 or to v. However, in the case where x_1 is connected to y_2, it is necessary that u connects to y_2 and x_2 to v, which after renaming the vertices is symmetric to the last case.

Now we treat the above cases individually. Note that w may not be a subdivision vertex, as otherwise we could contract its incident edges to the edge uv to obtain a smaller wrung obstruction. We will use this argument on several occasions.

Case I: Some of the vertices x_1, x_2, y_1 and y_2 coincide. Since $x_1 \neq x_2$ and $y_1 \neq y_2$ and because H has at least one vertex of degree 1, at most two of these vertices can coincide. By symmetry, we can assume that x_1 coincides with one of the vertices y_1 or y_2.

Case Ia: We have $x_1 = y_1$. The rotation schemes at u and v imply that x_2 and y_2 are embedded on the same side of the cycle C formed by ux_1vw, and therefore are embedded on different sides of C in $\mathcal{H}$. Hence, if G contains the edge x_2y_2, we obtain obstruction 1.

Now assume that x_2y_2 is not in G. Note that not both x_2 and y_2 can be subdivision vertices, as both would be part of a subdivision of the edge uv. Without loss of generality, we assume that x_2 is not a subdivision vertex. Assume first that G has the edge y_2u. Then x_2 must connect to two vertices in the set $\{x_1, v, w\}$. The embedding of the edge y_2u implies that x_2 cannot be adjacent to both w and x_1, and hence we either have edges x_2x_1 and x_2v or x_2w and x_2v. In both cases, the fact that $\{u, v\}$ is not a separator implies that the edge wx_1 is in G. The cycle C together with the edges uy_2, y_2v, ux_2, x_2v, and wx_1 then forms obstruction 4.

We can therefore assume that y_2u is not an edge of G, and hence y_2 is not a subdivision vertex. It follows that y_2 is adjacent to w and x_1. Planarity then implies that x_2 cannot be adjacent to v, so x_2 is adjacent to w and to x_1. To prevent $\{x_1, w\}$ from being a separator, G must contain the edge uv. The cycle C together with the edges x_1y_2, y_2w, x_1x_2, x_2w and uv again forms obstruction 4. This closes the case $x_1 = y_1$.

Case Ib: We have $x_1 = y_2$. We again distinguish two cases, based on whether one of x_2 or y_1 is a subdivision vertex.

Case Ib1: One of x_2 or y_1 is a subdivision vertex. Since both cases are symmetric we only treat the case that x_2 is a subdivision vertex. The only possibility is that x_2v is in G, otherwise x_2 would be part of a triangle.

We claim that G must contain x_1y_1 and y_1w. Note that neither of x_1, y_1 and w may be a subdivision vertex, as they would form a subdivision of the same edge uv, which is already represented by ux_2v. Now consider the graph consisting of H and the edge x_2v. The set uv is a separator of this graph with connected components x_1, y_1 and w. Since G is a subdivision of a 3-connected graph and neither of these vertices is a subdivision vertex G must contain two of the three possible edges between these vertices. If it contains both desired edges we are done. Otherwise G contains the edge x_1w. Planarity and the

presence of the edge x_2v requires that x_1w must be embedded on the same side of the cycle $C = ux_1vwu$ as y_1, thus enclosing y_1 in the triangle formed by x_1, v and w. Since y_1 has degree 3 in G it must be adjacent to x_1 and w, as claimed.

Since x_2 and y_1 are on distinct sides of C in $\mathfrak{R}^+$ they are embedded on the same side in $\mathcal{H}$. The graph consisting of the cycle C and the unprescribed edges ux_2, x_2v, x_1y_1 and y_1w forms obstruction 16.

Case Ib2: We have $x_1 = y_2$ and none of the vertices is a subdivision vertex. We make a case distinction, based on the adjacencies of x_2, which must be connected to at least two vertices from the set $\{x_1, v, w\}$. Note that similarly, y_1 must be connected to at least two of the vertices in $\{u, x_1, w\}$.

Assume first that x_2 is connected to all three of these vertices. If G contains the edge y_1u, then the cycle ux_1vwu together with the edges x_2x_1, x_2w, vy_1 and y_1u would form obstruction 16. Otherwise it must contain y_1x_1 and y_1w, in which case the cycle ux_1vwu together with edges ux_2, x_2v, y_1x_1 and y_1w forms obstruction 16. This concludes the case were x_2 is adjacent to all three vertices in $\{x_1, v, w\}$.

Next, we treat the case that x_2 is adjacent to x_1 and to w, but not to v. Now the set $\{x_1, w\}$ is a separator in the graph consisting of H and the edges constructed so far. Since G is 3-connected and planar, it must contain either uy_1 or x_2v. Since the latter is excluded, it contains uy_1. Again, the cycle and the edges uy_1, y_1v, x_1x_2 and x_2w form obstruction 16.

The remaining cases are that x_2 is not adjacent to one of x_1 and w. Since these cases are symmetric we only consider the case that x_2 is not adjacent to w, and thus G contains x_2x_1 and x_2v. As in case Ib1, we argue that G must contain x_1y_1 and y_1w, which can again be used to form obstruction 16. This concludes the treatment of the cases where x_1, x_2, y_1 and y_2 are not distinct.

Case II: The vertices x_1, x_2, y_1 and y_2 are distinct and one of the vertices is subdividing. Without loss of generality we assume that x_1 is subdividing, and we consider subcases based on the adjacencies of x_1.

Case IIa: The vertex y_1 is also subdividing, and G contains the edge x_1y_1. If x_2 was a subdivision vertex, it could not be adjacent to v, as the corresponding edge would be parallel to the edge subdivided by x_1. Therefore it would have to be adjacent to y_2, which would give obstruction 2, by contracting the path uwv to a single vertex. Hence, we can assume that x_2 is not subdividing, and by a symmetric argument also that y_2 is not subdividing. Hence, each of them needs degree 3 and since they are embedded on the same side of the cycle ux_1y_1vu, they must be adjacent, which again results in obstruction 2.

Case IIb: The vertex y_2 is also subdividing and G contains the edge x_1y_2. As in the previous case it can be seen that neither of x_2 and y_1 may be subdividing. Since they must be embedded on different sides of the cycle ux_1y_2vwu in $\mathfrak{R}^+$, the graph G must contain the edges x_2w, x_2v, y_1u and y_1w, in order for them to have degree 3. Altogether, this yields obstruction 12.

Case IIc: The vertex x_1 is subdividing and adjacent to v. Here, we consider several subcases based on which of the other vertices are subdividing.

Case IIc1: The vertex x_2 is also subdividing and adjacent to y_1. Again we can see that neither of y_1 and y_2 may be subdividing, as this would cause parallel edges. Hence they need degree 3. In particular, y_1 must be adjacent to two of the vertices u, w, y_2. However, the edge y_1u would be parallel to the subdivided edge ux_2y_1, and is thus excluded, and we have edges y_1y_2 and y_1w. Planarity and the degree of y_2 imply that also y_2w is in G. The constructed edges together with H form obstruction 9.

Case IIc2: The vertex x_2 is also subdividing and adjacent to y_2. Note that neither y_1 nor y_2 can be subdividing, as this would either cause parallel subdivided edges. Here, we distinguish two cases, based on how the edge x_1v is embedded. It can either be embedded so that y_1 and y_2 are on the same of on different sides of the cycle ux_1vwu.

In the latter case, since y_1 and y_2 both need degree 3, we obtain the edge y_1u, y_1w, y_2u and y_2w. Then H and the edges x_2y_2, y_2u, y_1u, and x_1v form obstruction 5.

Now assume that $y_1$and y_2 are embedded on the same side. In particular, y_1 is embedded on one side of the cycle $ux_1vy_2x_2u$, while all other vertices are on the cycle or embedded on the other side. Since the cycle only contains three non-subdividing vertices, namely u, v and y_2, y_1 must be connected to all of them, in particular G contains the edge y_1u. Further, w needs degree 3 and the only planar option is wy_2, which therefore also must be present. Then H and the edges x_1v, x_2y_2, y_2w and y_1u gives obstruction 5.

Case IIc3: The vertex x_1 is subdividing and adjacent to v, and additionally y_1 is also subdividing and adjacent to x_2. Again, none of the other vertices may be subdividing, because this would either cause parallel subdivided edges or would be symmetric to a previous case. The rotation schemes of u and v in $\mathfrak{R}^+$ imply a unique embedding of these edges, where x_2 and y_2 are on the same side of the cycle ux_1vwu. Both need degree 3, and the only attachments on the cycle are u, v and w. The edge x_2v would be parallel to the edge subdivided by y_1, hence x_2 is adjacent to w and y_2. Now, y_2 needs one more adjacency and by planarity the only option is the edge y_2w. Together H and the edges x_1v, y_1x_2, x_2y_2 and y_2w form obstruction 13.

Case IIc4: The vertex x_1 is subdividing and adjacent to v, and further y_2 is subdividing and adjacent to x_2. Again, none of the other vertices may be subdividing, because of parallel edges. The rotation schemes of u and v in $\mathfrak{R}^+$ allow two different embeddings. Either, y_1 and x_2 are embedded on the same or on different sides of the cycle ux_1vwu.

If they are embedded on different sides, then since y_1 needs degree 3, it must be adjacent to u and w. Further, x_2 needs another adjacency. By planarity, the only options are x_2w and x_2v. The latter would be parallel to the edge subdivided by y_2, and therefore $x_2w \in E(G)$. Now H and x_1v, x_2y_2, x_2w and y_1u form obstruction 5.

Now assume that y_1 and x_2 are embedded on the same side of ux_1vwu. Vertex w needs degree 3 and the only planar option is the edge wx_2. Further, y_1 must have degree 3 and the only possibly adjacent vertices are u and x_2, in particular $y_1u \in E(G)$. The graph H and the edges x_1v, x_2w, x_2y_2 and y_1u form obstruction 5.

Case IIc5: Finally, there is the case where x_1 is subdividing and adjacent to v, but no other vertex is subdividing. We distinguish cases, based on the embedding, in particular on the relative position of the vertices x_2, y_1 and y_2 with respect to the cycle $C = ux_1vwu$. It can either happen that all three of them are embedded on the same side of C, it can be that y_1 is embedded on one side and x_2 and y_2 on the other one, and finally it can be that y_1 and y_2 are on the same side and x_2 is on a different one.

We start with the latter case. Assume that x_2 is embedded on one side of C and x_2 and y_2 are on the other side. Since x_2 is not subdividing, x_2 must be adjacent to w and v. Further, y_1 and y_2 lie on the same side of C, and both need two attachments. Hence, the edge y_1y_2 must be in G. If y_1u is in G, then, since $\{u, v\}$ would form a separator otherwise, also y_1w must be in G, and for degree reasons of y_2 also the edge y_2u. Then the PEG contains obstruction 14. Assume that y_1u is not in G. Then y_1w must be in G since y_1 has degree at least 3. Further, since $\{w, v\}$ may not be a separator, we also find the edge y_2u, and thus again obstruction 14.

Next, assume that y_1 is on one side of the cycle C and x_2 and y_2 are on the other side.

Since y_1 has degree 3, it must in particular be adjacent to u. Further, since x_2 and y_2 both need degree 3 and are embedded on the same side of C, which only contains three non-subdividing vertices, they must also be adjacent. Finally, in the graph consisting of H and the edges constructed so far, the set $\{u, v\}$ forms a separator. Planarity and 3-connectivity of G imply that either x_2w or y_2w is in G. In either way, we obtain obstruction 5.

It remains to deal with the case where the vertices x_2, y_1 and y_2 are on the same side of C. We apply Lemma 7.6 with vertices u and v corresponding to u and v of the lemma and $u_1 = x_2, u_2 = w, v_1 = y_1$ and $v_2 = y_2$. We now treat the four cases that can arise in Lemma 7.6.

If case 1 of Lemma 7.6 applies, we obtain edges vx_2, x_2w, wy_1, y_1y_2 and y_2u. However, with the fixed embedding of x_1v, it can be seen that y_1w and y_2u cannot be added in a planar way, so this case cannot occur.

If case 2 of the lemma applies, we obtain edges uy_1, y_1y_2, x_2y_2, x_2w. Now, H and the edges x_2y_2, x_2w, x_1v and uy_1 form obstruction 5.

If case 3 of the lemma applies, we obtain a vertex $w' \neq w, y_2, u, v$ with paths $w' \to w \to y_2$, $w' \to y_1$ and $w' \to x_2$. Since G has only seven vertices, w' must coincide with x_2 or y_1. Suppose that $w' = x_2$. If $y_1y_2 \in E(G)$, we obtain obstruction 9. Hence, assume that $y_1y_2 \notin E(G)$. This implies that $x_2y_2 \in E(G)$ since y_2 has degree 3 in G. Then H with edges x_2y_1, x_2y_2, y_2w and x_1v forms obstruction 13. This closes the case $w' = x_2$.

Now suppose that $w' = y_1$. Thus, in addition to the edge x_1v,we have edges y_1w, wy_2 and y_1x_2. The facts that y_2 has degree 3 in G and that G is planar, with the embedding of x_1v already fixed, implies $y_1y_2 \in E(G)$. Now H together with the constructed edges, except for wy_1, forms obstruction 9.

Now, assume that case 4 of the lemma applies. In addition to the already embedded edge x_1v, we obtain a vertex $w' \neq w, y_2, u, v$ and paths $w' \to y_2 \to w$, $w' \to y_1$ and $w' \to x_2$. Again, w' coincides with x_2 or y_1. If $w' = x_2$, this forms obstruction 13, and if $w' = y_1$, it forms obstruction 9.

Case IId: The vertex x_1 is subdividing and adjacent to y_1, but y_1 is not subdividing. We distinguish subcases based on whether other vertices are subdividing.

Case IId1: The vertex x_2 is subdividing and adjacent to y_2, then by contracting the path uwv to a single vertex, we obtain obstruction 2.

Case IId2: The vertex y_2 is subdividing and adjacent to x_2. As in the previous case, we obtain obstruction 2. As argued in the description of the case analysis, this covers all instances, where a vertex besides x_1 is subdividing.

Case IId3: The vertex x_1 is subdividing and adjacent to y_1, and no other vertex is subdividing. Clearly, if $x_2y_2 \in G$, we obtain obstruction 2. We can therefore assume that this is not the case. Hence, x_2 must be adjacent to at least two vertices in the set $\{w, v, y_1\}$.

First, assume that x_2w and x_2v are in G. The edge x_2v admits two distinct embeddings, however in one of them the cycle x_2wvx_2 would enclose only the vertex y_2, which would imply the existence of the excluded edge x_2y_2. We can therefore assume that all vertices that do not belong to the cycle are on the same side of it. Since y_2 has degree 3 in G, this implies the existence of the edges y_1y_2 and y_2u. Since $\{u, v\}$ still would form a separating pair, the edge wy_1 must also be present. Omitting the edge x_2w yields obstruction 5.

Next, assume that x_2w and x_2y_1 are in G. Since y_2 has degree 3, and y_1y_2 is excluded, G must contain the edges y_1y_2 and y_2w. Until now $\{w, y_1\}$ would still form a separating pair. Since x_2y_2 is excluded G contains the edge uv. Altogether, this is obstruction 15.

Finally, assume that x_2v and x_2y_1 are in G. Again, only one of the embeddings of x_2v does not force the edge x_2y_2 to be present. Since y_2 has degree 3, it must be adjacent to u

and w. As $\{u, v\}$ would still be a separating pair, we also obtain the edge wy_1. Omitting the edges y_2w and x_2y_1 yields obstruction 5. This finishes the case where x_1 is subdividing and adjacent to y_1.

Case IIe: The vertex x_1 is subdividing and adjacent to y_2, which is not subdividing. No other vertex may be subdividing, as this would be symmetric to case IIc. Clearly, y_1 must be adjacent to two vertices in the set $\{u, w, y_2\}$. We consider several cases.

Assume that y_1 is adjacent to y_2 and w. If G contains x_2v, this forms obstruction 9. Therefore, assume that x_2v is not in G. It follows that x_2 is adjacent to w and y_2. Since $\{w, y_2\}$ still forms a separator, we also get edge y_1u. By omitting y_1y_2 and y_1w, we obtain obstruction 13.

Next, assume that y_1 is adjacent to y_2 and u but not w. Then x_2 must be adjacent to y_2, otherwise uv would form a separator or x_2 would have degree 2. Also, w must be adjacent to x_2, otherwise x_2 or w would have degree 2. This yields obstruction 13.

Finally, assume that y_1 is adjacent to u and to w, and not to y_2 (otherwise one of the previous cases would apply). Clearly, x_2 must be adjacent to two of the three vertices w, v and y_2. It is not possible that x_2 is only adjacent to w and v, since $\{u, v\}$ would still form a separating pair. Hence $x_2y_2 \in E(G)$. Also x_2 must be adjacent to w, otherwise uv would be a separator or x_2 would have degree 2. This gives obstruction 13.

This closes the case where x_1 is subdividing and adjacent to y_2, and thus all cases where G has a subdividing vertex.

Case III: The graph G does not have any subdividing vertices, and thus is 3-connected. We distinguish two subcases, based on whether G contains the edge uv.

Case IIIa: The graph G is 3-connected and contains the edge uv. The edge uv can be embedded in a variety of different ways in $\mathfrak{R}^+$. We distinguish cases, based on this embedding, in particular, the relative position of the cycle $C = uwvu$ and the vertices x_1, x_2, y_1 and y_2.

Case IIIa1: First, assume that all these vertices are embedded on the same side of C. We apply Lemma 7.6 on the vertices u and v with $u_i = x_i$ and $v_i = y_i$ for $i = 1, 2$. Suppose that case 1 of the lemma applies. Then we obtain a simple path $v \to x_1 \to x_2 \to y_1 \to y_2 \to u$. If w is not contained in any of the subpaths, we can contract wv and obtain obstruction 7. Further, by planarity, w cannot subdivide any of the subpaths $x_1 \to x_2, y_1 \to y_2$ or $x_2 \to y_2$. Hence, it must subdivide $x_1 \to v$ or $y_2 \to u$, which is completely symmetric, and we assume without loss of generality that w subdivides the path $x_1 \to v$ and all other subpaths consist of a single edge, each. Again, contracting wv yields obstruction 7.

Case 2 of the lemma is completely symmetric, we therefore continue with case 3. We obtain a vertex $w' \neq x_2, y_2, u, v$ and paths $w' \to x_2 \to y_2, w' \to x_1, w' \to y_1$. Further, w may subdivide one of these paths. First of all, w' must coincide with either of x_1, y_1 or w.

If $w' = x_1$, we obtain obstruction 2, unless w subdivides the subpath $x_1 \to y_1$ (or symmetrically $x_2 \to y_2$). We therefore assume that w subdivides x_1y_1, and thus all other paths consist of single edges. Since y_1 must have degree at least 3, at least one of the three edges y_1y_2, y_1x_2 or y_1x_1 must be present, resulting in obstructions 8, 15 and 2, respectively.

If $w' = y_1$, we again obtain obstruction 2, unless w subdivides either $x_1 \to y_1$ or $x_2 \to y_2$. Again these cases are symmetric, and we assume w subdivides $x_1 \to y_1$. Since x_1 has degree 3, it is either adjacent to y_1 or to x_2. This leads to obstructions 2 and 15, respectively.

If $w' = w$, x_1x_2 must be in G because of planarity and since x_1 has degree at least 3. Further, y_1 has degree at least 3 and therefore G either contains y_1y_2 or y_1x_2, which yields obstructions 8 and 15, respectively.

Since case 4 of the Lemma 7.6 is completely symmetric, this closes the case where the

cycle $C = uwvu$ bounds an empty triangle.

Case IIIa2: The graph G is 3-connected, contains the edge uv and the cycle $C = uwvu$ is embedded such that it separates x_1 from the vertices x_2, y_1 and y_2. In this case, x_1 must be adjacent to v and w in G. We apply Lemma 7.6 to vertices u and v with $u_1 = x_2, u_2 = w, v_1 = y_1$ and $v_2 = y_2$.

In case 1 of the lemma, we obtain a path $v \to x_2 \to w \to y_1 \to y_2 \to u$, which cannot be embedded in a planar way into G, so this case does not occur.

In case 2 of the lemma, we obtain edges $uy_1, y_1y_2, y_2x_2, x_2w$ and wv. Together with the edge x_1v, this forms an instance of obstruction 5.

In case 3 of the lemma, we obtain a vertex $w' \neq w$ paths $w' \to w \to y_2$, $w' \to x_2$ and $w' \to y_1$. The vertex x_1 cannot subdivide any of these paths and also $w' = x_1$ is not possible by planarity.

Suppose that $w' = x_2$, we thus obtain edges x_2w, y_2w and x_2y_1. We already know that x_1v is an edge of G. Further, y_2 must have another adjacency, which can either be y_1 or x_2. In the former case, we find obstruction 9, in the latter obstruction 13.

Suppose that $w' = y_1$, we then have edges x_2y_1, y_2w and y_1w. Planarity and the degree of y_2 imply that y_1y_2 is in G, and as above x_1v is in G. Together, this yields obstruction 9.

In case 4 of the lemma, we obtain a vertex $w' \neq y_2$ and paths $w' \to y_2 \to w, w' \to y_1$ and $w' \to x_2$. Again, by planarity, x_1 may neither coincide with w' nor subdivide any of the paths.

Suppose that $w' = x_2$, and we thus have edges x_2y_2, y_2w and x_2y_1. As above we find the edge x_1v. This gives obstruction 13.

Suppose that $w' = y_1$. This yields edges x_2y_1, y_2w and y_1y_2. As above we find edge x_1v, and thus obstruction 9. This closes the case where exactly one of the vertices is enclosed by the cycle $uwvu$.

Case IIIa3: The edge uv is embedded such that the cycle $C = uwvu$ separates x_1 and x_2 from y_1 and y_2. We may assume that x_1 and x_2 are to the right of the directed cycle $u \to v \to w \to u$, while y_1 and y_2 are to its left. Note that in this case, the vertices x_1 and x_2 must be adjacent, because otherwise both of them would have to be adjacent to u, v and w, contradicting planarity. By 3-connectedness, both v and w must be adjacent to at least one of x_1 and x_2, and each x_i must be adjacent to at least one of v and w. Planarity implies that x_1w and x_2v must both be edges of G. An analogous argument for y_1 and y_2 implies that uy_1, y_1y_2 and y_2w are all edges of G. This forms obstruction 11.

Case IIIa4: The edge uv is embedded such that the cycle $C = uwvu$ separates x_1 and y_1 from x_2 and y_2. Clearly, x_1 and y_1 both need degree 3. However, C has only vertices u, w and v, and not both x_1 and y_1 can be connected to all vertices of C in a planar way (otherwise we could find a planar drawing of $K_{3,3}$). Hence, G must contain the edge x_1y_1, and by a symmetric argument also x_2, y_2, which results in obstruction 2.

Case IIIb: The graph G is 3-connected and does not contain the edge uv. Since G is 3-connected it contains three vertex-disjoint paths p_1, p_2 and p_3 from $\{x_1, u, x_2\}$ to $\{y_1, v, y_2\}$. We distinguish cases, based on which endpoints are connected by the paths.

Case IIIb1: The path p_1 connects x_1 to y_1 and p_2 connects u to v. Clearly, p_3 must then connect x_2 to y_2. Since G does not contain the edge uv, p_2 must contain the vertex w, which implies that p_1 and p_3 consist of a single edge, each. This yields obstruction 2.

Case IIIb2: The path p_1 connects x_1 to y_1 and p_2 connects u to y_2. Clearly, p_3 must then connect y_2 to v. We distinguish cases, based on whether w is contained in one of these paths.

First, suppose that none of these paths contains w, that is each of them consists of a

single edge. The edges x_2v and y_2u admit two different embeddings that are completely symmetric. We therefore assume that $uwvy_2u$ bounds a face in the graph consisting of H and the paths $p_i, i = 1, 2, 3$ with the embedding inherited from $\mathfrak{R}^+$. Then the cycle x_2vwux_2 separates y_2 from x_1 and y_1. Since y_2 needs degree at least 3, we either have x_2y_2 or y_2w in G. The former would yield obstruction 2, thus we assume the latter. However, still $\{u, v\}$ would form a separating pair, thus implying that w needs to be adjacent to either x_1 or to y_1. In both cases we obtain obstruction 5. Hence, we can assume that w is contained in one of the paths.

First, assume that w is contained in p_1. Again, the embedding choices offered by the edges x_2v and y_2u are completely symmetric, and as above we assume that the cycle x_2vwux_2 separates y_2 from x_1 and y_1. If x_1y_1 was in G, we could replace the path p_1 by this edge and the previous case would apply; we therefore assume that this is not the case. If x_1v was in G, then the fact that y_1 needs another adjacency would again imply that x_1y_1 is in G. Since x_1 needs one more adjacency, the only option is the edge x_1x_2. Similarly, for y_1 the only option is the edge y_1x_2. The graph then contains obstruction 9.

Second, assume that w is contained in p_2. We then have edges x_1y_1, y_2w and x_2v, which have a unique embedding, in which the cycle x_2vwux_2 separates y_2 from x_1 and y_1. Since y_2 needs degree at least 3, it must be adjacent to either x_2 or to u. In the former case, we again get obstruction 2. In the latter case, we can replace the path p_2 with the edge uy_2 and we are in the case that w is not contained in any of the three paths.

Finally, the case that w is contained in p_3 is completely symmetric to the previous case. This closes the case, where p_1 connects x_1 to y_1 and p_2 connects u to y_2.

Case IIIb3: The path p_1 connects x_1 to v, p_2 connects u to y_2, and thus p_3 connects x_2 to y_1. Again, we consider cases based on whether w is contained in one of these paths. First, suppose that w is not contained in any of these paths. Then we have edges x_2y_1, x_1v and y_2u, which have a unique planar embedding in $\mathfrak{R}^+$. Since w has degree 3 it must be connected to x_1 or y_2. Both are completely symmetric, and we assume wy_2 is in G. So far, the set $\{u, v\}$ would still form a separating pair. Hence G contains at least one of the edges y_1y_2, y_2x_2 and wx_1, resulting in obstruction 9, 13 and 12, respectively.

We can therefore assume that w subdivides one of the paths. Suppose that w subdivides p_1, that is, we have edges x_1w, uy_2 and x_2y_1. Clearly, x_1 must be adjacent to one of x_2, v and y_1. However, x_1x_2 produces obstruction 9, x_1y_1 produces obstruction 13, and if x_1v is in G, we replace the path p_2 by x_1v and are in the case where w is not contained in any of the three paths.

The case that w is contained in p_2 is completely symmetric, we therefore assume it is contained in p_3, and we thus have edges x_1v, y_2u, x_2w and y_1w. There are two possible planar embeddings. One, in which x_1v comes after vw and before vy_2 in counter-clockwise direction around v, and one in which x_1v comes after vy_2 and before vy_1 in counter-clockwise direction.

We start with the first case. Suppose that G contains the edge x_1y_2. The vertex x_2 needs another adjacency, which can either be x_1 or v. Analogously, y_1 needs to be adjacent to either u or y_2. The combination x_1x_2 and y_1y_2 gives obstruction 11, any of the combination x_2v, y_1y_2 and x_1x_2, y_1u gives obstruction 9, and the combination x_2v and y_2u gives obstruction 12.

Now assume that G does not contain the edge x_1y_2. Then x_1 needs to be adjacent to x_2; its only options are x_2 and v, but in the latter case x_2 still needs an edge and the only possibility is x_1. Analogously, y_1y_2 must be in G. Together this forms obstruction 11.

Now suppose that the second embedding applies, and x_1v comes after vy_2 and before vy_1

in counter-clockwise direction around v. If G contains x_1y_1, then it cannot contain x_2y_2 as well, otherwise we would get obstruction 2. Hence, x_2 must be adjacent to v. Since y_2 is not adjacent to x_2, it must be adjacent to x_1, and we thus obtain obstruction 13. We can hence assume that G does not contain x_1y_1, and symmetrically also not x_2y_2. We hence get edges x_2v and y_1u. But then still $\{u, v\}$ forms a separator, which shows that either x_1y_1 or x_2y_2 is in G; a contradiction. This finishes the last case, and thus concludes the proof. □

$\mathfrak{R}$ has an edge-compatible embedding

Assume now that the embedding $\mathfrak{R}^+$ of the skeleton $\mathfrak{R}$ is edge-compatible but not cycle-compatible. We first give a sketch of our general proof strategy. Our analysis of this situation strongly relies on the concept of C-bridges, which has been previously used by Juvan and Mohar in the study of embedding extensions on surfaces of higher genus [JM05], and which is also employed (under the term *fragment*) by Demoucron, Malgrange and Pertuiset in their planarity algorithm [DMP64].

Let F be a graph and C a cycle of F. A *C-bridge* is either a chord of C or a connected component of $F - C$, together with all vertices that connect it to C, which are known as *attachments* of the bridge. A vertex of C that is an endpoint of an edge of a C-bridge X is called an *attachment* of X. Let $\mathrm{att}(X)$ denote the set of attachments of X. A bridge that consists of a single edge is *trivial*. We will mostly be interested in bridges formed in the graph $\mathfrak{R}$ with respect to cycles that are projections of cycles of H. Notice that in this case any nontrivial bridge in $\mathfrak{R}$ has at least three attachments, because $\mathfrak{R}$ is 3-connected. It is not hard to verify that in a 3-connected graph, once we select the position of a single C-bridge with respect to C, the positions of all the remaining C-bridges are uniquely determined.

In our argument, we focus on cycles in $\mathfrak{R}$ that are projections of cycles in H. If $\mathfrak{R}^+$ violates cycle-compatibility, it means that H must contain a cycle C' that projects to a cycle C of $\mathfrak{R}$, and $\mathfrak{R}^+$ has a C-bridge that is embedded on the 'wrong' side of C. We concentrate on the substructures that enforce such 'wrong' position for a given C-bridge, and use them to locate planarity obstructions.

Let us describe the argument in more detail. Suppose again that C' is a cycle of H that projects to a cycle C of $\mathfrak{R}$. Let x be a vertex of H that does not belong to any $\mathfrak{R}$-edge belonging to C. We say that x is *happy* with C', if its embedding in $\mathfrak{R}^+$ does not violate cycle-compatibility with respect to the cycle C', that is, x is to the left of C' in $\mathcal{H}$ if and only if x is to the left of C in $\mathfrak{R}^+$. Otherwise we say that x is unhappy with C'. We say that a C-bridge B of $\mathfrak{R}$ is happy with C' if there is a vertex x happy with C' that projects into B, and similarly for unhappy bridges. A C-bridge that is neither happy nor unhappy is *indifferent*.

In our analysis of cycle-incompatible skeletons, we establish the following facts.

- With C and C' as above, if a single C-bridge is both happy and unhappy with C', then $(G, H, \mathcal{H})$ contains obstruction 1 or 4 (Lemma 7.16).
- Let us say that the cycle C' is *happy* if at least one C-bridge is happy with C', and it is *unhappy* if at least one C-bridge is unhappy with C'. If C' is both happy and unhappy, then $(G, H, \mathcal{H})$ contains obstruction 4, obstruction 16, or an alternating-chain obstruction (Lemma 7.17).
- Assume that the situation described above does not arise. Assume further that C' is an unhappy cycle of $\mathcal{H}$. Then any edge of H incident to a vertex of C' must project

into an $\mathfrak{R}$-edge belonging to C, unless $(G, H, \mathcal{H})$ contains obstruction 3 or one of the obstructions from the previous item. Note that this implies, in particular, that the vertices of C impose no edge-compatibility constraints (Lemma 7.19).

- If C_1' and C_2' are two facial cycles of $\mathcal{H}$ whose projection is the same cycle C of $\mathfrak{R}$, then any C-bridge is happy with C_1' if and only if it is happy with C_2', unless the graph G is non-planar, or the PEG $(G, H, \mathcal{H})$ contains obstruction 1 (Lemma 7.20).
- If H contains a happy facial cycle as well as an unhappy one, we obtain obstruction 18 (Lemma 7.21).
- If H contains an unhappy facial cycle, and if at least one vertex of $\mathfrak{R}$ imposes any non-trivial edge-compatibility constraints, then $(G, H, \mathcal{H})$ contains one of the obstructions 19–22 (Lemma 7.22).

Moreover, in each of the above claims, the corresponding obstructions can be found efficiently. Note that these facts guarantee that if $(G, H, \mathcal{H})$ is obstruction-free then $\mathfrak{R}$ has a compatible embedding. To see this, assume that $\mathfrak{R}^+$ is an edge-compatible but not cycle-compatible embedding of $\mathfrak{R}$. This means that at least one facial cycle of $\mathcal{H}$ is unhappy. This in turn implies that no cycle may be happy, and no vertex of $\mathfrak{R}$ may impose any edge-compatibility restrictions. Consequently, the embedding $\mathfrak{R}^-$ is compatible. Before we can prove the above claims, we first need some technical machinery, in particular the conflict graph of C-bridges and its properties.

Conflict graph of a cycle and minimality of alternating chains. For a cycle C and two distinct vertices x and y of C, an *arc* of C with endvertices x and y is a path in C connecting x to y. Any two distinct vertices of a cycle determine two arcs. Let u, v, x, y be four distinct vertices of a cycle C. We say that the pair $\{x, y\}$ *alternates with* $\{u, v\}$ if each arc determined by x and y contains exactly one of the two vertices $\{u, v\}$. If U and X are sets of vertices of a cycle C, we say that X *alternates* with U if there are two pairs of vertices $\{u, v\} \subseteq U$ and $\{x, y\} \subseteq X$ that alternate with each other.

Let now F be a graph containing a cycle C. Intuitively, a bridge represents a subgraph whose internal vertices and edges must all be embedded on the same side of C in any embedding of F. Thus, a C-bridge may be embedded in two possible positions relative to C. Moreover, if two bridges B_1 and B_2 have three common attachments, or if the attachments of B_1 alternate with the attachments of B_2, then in any planar embedding, B_1 and B_2 must appear on different sides of C. This motivates the definition of two types of conflicts between bridges. We say that two C-bridges X and Y of F have a *three-vertex conflict* if they share at least three common attachments, and they have a *four-vertex conflict* if $\mathrm{att}(X)$ alternates with $\mathrm{att}(Y)$. Two C-bridges have a conflict if they have a three-vertex conflict or a four-vertex conflict. This gives rise to a conflict graph of F with respect to C. For a cycle C, define the *conflict graph* K_C to be the graph whose vertices are the C-bridges, and two vertices are connected by an edge of K_C if and only if the corresponding bridges conflict. Define the *reduced conflict graph* K_C^- to be the graph whose vertices are bridges of C, and two bridges are connected by an edge if they have a four-vertex conflict.

As a preparation, we first derive some basic properties of conflict graphs.

Lemma 7.9. *If F is a planar graph, then for any cycle C of F the conflict graph K_C is bipartite (and hence K_C^- is bipartite as well).*

Proof. In any embedding of F, each C-bridge must be completely embedded on a single side of C. Two conflicting bridges cannot be embedded on the same side of C. □

Consider now the situation when C is a cycle of length at least 4 in a 3-connected graph F. The goal is to show that in this case also the reduced conflict graph K_C^- is connected. To prove this we need some auxiliary lemmas. The first states that if the attachments of a set of bridges alternate with two given vertices x and y of C, then the set must contain a C-bridge whose attachments alternate with x and y, provided that the set of bridges is connected in the reduced conflict graph K_C^-.

Lemma 7.10. *Let F be a graph and let C be a cycle in F. Let K be a connected subgraph of the reduced conflict graph K_C^- and let att(K) be the set of all attachment vertices of the C-bridges in K, that is, att$(K) = \bigcup_{X \in K}$ att(X). If $\{x, y\}$ is a pair of vertices of C that alternates with att(K), then there is a bridge $X \in K$ such that the pair $\{x, y\}$ alternates with att(X).*

Proof. Let α and β be the two arcs of C with endvertices x and y. Let K_α be the set of C-bridges from K whose all attachments belong to α, and let K_β be the set of bridges from K with all their attachments in β. Note that both K_α and K_β are proper subsets of K, because $\{x, y\}$ alternates with att(K).

Since no bridge in K_α conflicts with any bridge in K_β, and since K is a connected subgraph in the reduced conflict graph, there must exist a bridge $X \in K$ that belongs to $K \setminus (K_\alpha \cup K_\beta)$. Clearly, X has at least one attachment in the interior of α as well as at least one attachment in the interior of β. Thus, att(X) alternates with $\{x, y\}$. □

Next, we show that in a 3-connected graph, unless C is a triangle, its reduced conflict graph K_C^- is connected.

Lemma 7.11. *Let C be a cycle of length at least 4 in a 3-connected graph F. Then the reduced conflict graph K_C^- is connected (and hence K_C is connected as well).*

Proof. We first show that for a cycle C of length at least 4 and a set of C-bridges K that form a connected component in K_C^-, every vertex of C is an attachment of at least one bridge in K.

Claim 7.3. Let C be a cycle of length at least four in a 3-connected graph F. Let K be a connected component of the graph K_C^-, and let att(K) be the set $\bigcup_{X \in K}$ att(X). Then each vertex of C belongs to att(K).

Suppose that some vertices of C do not belong to att(K). Then there is an arc α of C of length at least 2, whose endvertices belong to att(K), but none of its internal vertices belongs to att(K). Let x and y be the endvertices of α. Let β be the other arc determined by x and y. Notice that β also has length at least 2.

Since F is 3-connected, $F - \{x, y\}$ is connected, and in particular, there is a C-bridge Y that has at least one attachment u in the interior of the arc α and at least one attachment v in the interior of β. Clearly $Y \notin K$, since Y has an attachment in the interior of α.

Since the pair $\{u, v\}$ alternates with $\{x, y\} \subseteq$ att(K), Claim 7.10 shows that there is a bridge $X \in K$ whose attachments alternate with $\{u, v\}$. Then X and Y have a four-vertex conflict, which is impossible because K is a connected component of K_C^- not containing Y, and the claim holds.

We are now ready to prove the lemma. Let K and K' be two distinct connected components of K_C^-. Choose a bridge $X \in K$. Let u and v be any two attachments of X that are not connected by an edge of C. By Claim 7.3, each vertex of C is in att(K'), so att(K') alternates with $\{u, v\}$, and hence by Claim 7.10, the set K' has a bridge Y whose

attachments alternate with the attachments of X. Hence, X and Y have a four-vertex conflict and belong to the same connected component in K_C^-. □

Next, we show that if we have an induced path in the conflict graph, then we can find a corresponding sequence of bridges and pairs of their attachment vertices such that consecutive pairs alternate. This lemma will be the main working tool for extracting alternating chains from non-planar PEGs.

Lemma 7.12. *Let C be a cycle of length at least 4 in a 3-connected graph F and let P be an induced path with $k \geq 2$ vertices in the graph K_C^-. Let $X_1, X_2, \ldots, X_k$ be the vertices of P, with X_i adjacent to X_{i+1} for each $i = 1, \ldots, k-1$. Then for each $i \in \{1, \ldots, k\}$ we may choose a pair of vertices $\{x_i, y_i\} \subseteq att(X_i)$, such that for each $i = 1, \ldots, k-1$ the pair $\{x_i, y_i\}$ alternates with the pair $\{x_{i+1}, y_{i+1}\}$.*

Proof. For each $j \leq k$, select a set $S_j \subseteq \mathrm{att}(X_j)$ in such a way that for each $i < k$ the set S_i alternates with S_{i+1}. Such a selection is possible, for example, by taking $S_j = \mathrm{att}(X_j)$. Assume now that we have selected $\{S_j, j = 1, \ldots, k\}$ so that their total size $\sum_{j \leq k} |S_j|$ is as small as possible. We claim that each set S_j consists of a pair of vertices $\{x_j, y_j\}$.

Assume for contradiction that this is not the case. Since obviously each S_j has at least two vertices, assume that for some j we have $|S_j| \geq 3$. Clearly, this is only possible for $1 < j < k$. Select a pair of vertices $\{x_{j-1}, y_{j-1}\} \subseteq S_{j-1}$ and a pair of vertices $\{x_{j+1}, y_{j+1}\} \subseteq S_{j+1}$ such that both these pairs alternate with S_j. Since the sets S_{j-1} and S_{j+1} do not alternate, there is an arc α of C with endvertices $\{x_{j-1}, y_{j-1}\}$ that has no vertex from S_{j+1} in its interior, and similarly there is an arc β with endvertices $\{x_{j+1}, y_{j+1}\}$ and no vertex of S_{j-1} in its interior.

Since both $\{x_{j-1}, y_{j-1}\}$ and $\{x_{j+1}, y_{j+1}\}$ alternate with S_j, there must be a vertex $x_j \in S_j$ that belongs to the interior of α, and a vertex $y_j \in S_j$ belonging to the interior of β. The pair $\{x_j, y_j\}$ alternates with both S_{j-1} and S_{j+1}, contradicting the minimality of our choice of S_j. □

Our next goal is to link the conflict graph with the elements of Ach_k. Note that for all elements of Ach_k, the conflict graph forms a path of length k. To establish a link, we consider pairs of a graph and a cycle such that the conflict graph forms a path. Let F be a graph, and let C be a cycle in F. We say that the pair (F, C) forms a *conflict path*, if each C-bridge of F has exactly two attachments and the conflict graph K_C is a path. (Note that if each C-bridge has two attachments, then the conflict graph is equal to the reduced conflict graph.)

By definition, every element of Ach_k forms a conflict path. The converse, is however not true. Suppose that (F, C) forms a conflict path. Let $e = uv$ be an edge of C. The edge e is called *shrinkable* if no C-bridge attached to u conflicts with any C-bridge attached to v. (The idea is that a shrinkable edge may be contracted without modifying the conflict graph.)

Before we can show that the elements of Ach_k are minimal non-planar PEGs, we first need a more technical lemma about conflict paths.

Lemma 7.13. *Assume that (F, C) forms a conflict path. Then each vertex of C is an attachment for at most two C-bridges.*

Proof. Suppose that (F, C) forms a conflict path, and a vertex $v \in C$ is an attachment of three distinct bridges X, Y and Z. These three bridges do not alternate, so there must be at least five bridges to form a path in K_C. Let x, y and z be the attachments of X, Y and

Z different from v. The three vertices x, y and z must be all distinct, because a pair of bridges with the same attachments would share the same neighbors in the conflict graph, which is impossible if the conflict graph is a path with at least five vertices.

Choose an orientation of C and assume that the four attachments appear in the order (v, x, y, z) with respect to this orientation. Let α_{vx}, α_{xy}, α_{yz}, and α_{zx} be the four internally disjoint arcs of C determined by consecutive pairs of these four attachments.

Let P_{xz} be the subpath of K_C that connects X to Z. At least one bridge of P_{xz} must have an attachment in the interior of α_{vx} and at least one bridge of P_{xz} must have an attachment in the interior of α_{zv}. So the attachments of Y alternate with the attachments of P_{xz} and by Lemma 7.10, at least one bridge in P_{xz} conflicts with Y. This means that Y is an internal vertex of P_{xz}.

Consider now the graph $P_{xz} - Y$. It consists of two disjoint paths P_x and P_z containing X and Z respectively. P_x has a vertex adjacent to X as well as a vertex adjacent to Y, but no vertex adjacent to Z. In particular, P_x has at least one attachment in the interior of α_{vx} and at least one attachment in the interior of α_{yz}. Similarly, P_z has an attachment in the interior of α_{xy} and in the interior of α_{zv}. Hence, the attachments of P_x alternate with the attachments of P_z. Using Lemma 7.10, we easily deduce that at least one bridge of P_x must conflict with a bridge of P_z, which is a contradiction. □

Next, we show that the structure of the attachment vertices on the cycle C of a conflict path (F, C) without shrinkable edges has a structure very similar to that of an alternating chain.

Lemma 7.14. *Assume that (F, C) forms a conflict path with $k \geq 4$ C-bridges. Let $X_1, \ldots, X_k$ be the C-bridges, listed in the order in which they appear on the path K_C. Let $\{x_i, y_i\}$ be the two attachments of X_i. Assume that C has no shrinkable edge. Then*

1. *The two attachments $\{x_1, y_1\}$ of X_1 determine an arc α_1 of length 2, and the unique internal vertex z_1 of this arc is an attachment of X_2 and no other bridge.*
2. *The two attachments $\{x_k, y_k\}$ of X_k determine an arc α_k of length 2 different form α_1, and the unique internal vertex z_k of this arc is an attachment of X_{k-1} and no other bridge.*
3. *All the vertices of C other than z_1 and z_k are attachments of exactly two bridges.*

Proof. We know from Lemma 7.13 that no vertex of C is an attachment of more than two bridges.

Let α and β be the two arcs of C determined by $\{x_1, y_1\}$. The bridges $X_3, \ldots, X_k$ do not alternate with X_1, so all their attachments belong to one of the two arcs, say β. The arc α then has only one attachment z_1 in its interior, and this attachment belongs to X_2 and no other bridge. It follows that α has only one internal vertex. This proves the first claim; the second claim follows analogously.

To prove the third claim, note first that any vertex of C must be an attachment of at least one bridge. Suppose that there is a vertex v that is an attachment of only one bridge X_j. Let u and w be the neighbors of v on C. By assumption, both u and w are attachments of at least one bridge that conflicts with X_j.

Assume first, that a single bridge Y conflicting with X_j is attached to both u and w. Since the arc determined by u and w and containing v does not have any other attachment in the interior, this means that Y conflicts only with the bridge X_j. Then $Y \in \{X_1, X_k\}$ and $v \in \{z_1, z_k\}$. Next, assume that the bridge X_{j-1} is attached to u but not to w, and

the bridge X_{j+1} is attached to w but not u. We then easily conclude that X_{j-1} conflicts with X_{j+1}, which is a contradiction. □

This directly implies that non-planar PEGs that form a conflict path and do not have shrinkable edges are k-fold alternating chains.

Corollary 7.1. *Let $(G, H, \mathcal{H})$ be a non-planar* PEG *for which H consists of a single cycle C of length at least 4 and two additional vertices u and v that do not belong to C, such that (G, C) forms a conflict path with bridges $X_1, \ldots, X_k$ along the path, each with attachments $\{x_i, y_i\}$. Let further X_i consist of the single edge $x_i y_i$ for $i = 2, \ldots, k-1$ and let X_1 consist of $x_1 u y_1$ and X_k of $x_k v y_k$. If C does not contain shrinkable edges then $(G, H, \mathcal{H})$ is an element of Ach_k.*

Proof. The non-planarity of G implies that u and v must be embedded on different sides of C if k is even, and on the same side if k is odd.

Clearly, the graphs G and H have the same vertex set. By assumption, each bridge X_i forms a path P_i, which satisfy the properties for k-fold alternating chains; they have the right lengths and contain the right vertices. Further, since (G, C) forms a conflict path their endpoints alternate in the required way.

Finally, as C has no shrinkable edges, Lemma 7.14 implies that all vertices of C have degree 4, with the exception of one of the attachments of X_2 and X_{k-1}, which have degree 3. This also implies that the length of the cycle is $k+1$, and thus $(G, H, \mathcal{H})$ thus is an element of Ach_k. □

We now employ the observations we made so far to show that every element of Ach_k is indeed a minimal obstruction.

Lemma 7.15. *For each $k \geq 3$, every element of Ach_k is a minimal obstruction.*

Proof. As observed before, Ach_3 contains a single element, which is the obstruction 4. Assume $k \geq 4$, and choose $(G', H', \mathcal{H}') \in \mathrm{Ach}_k$. Let C be the unique cycle of H, and let u and v be the two isolated vertices of H. Observing that $(G', H', \mathcal{H}')$ is not planar is quite straightforward: since no two conflicting bridges can be embedded into the same region of C, all the odd bridges $X_1, X_3, X_5, \ldots$ must be in one region while all the even bridges must be in the other region, and this guarantees that u or v will be on the wrong side of C.

Let us prove that $(G', H', \mathcal{H}')$ is minimal non-planar. The least obvious part is to show that contracting an edge of a cycle C always gives a planar PEG. If the cycle C contained a shrinkable edge $e = xy$, we might contract the edge into a single vertex x_e. After the contraction, the new graph still forms a conflict path, but the vertex x_e is an attachment of at least three bridges, which contradicts Lemma 7.13. We conclude that C has no shrinkable edge.

By contracting a non-shrinkable edge C, we obtain a new PEG $(G'', H'', \mathcal{H}'')$ where H consists of a cycle C' and two isolated vertices. The conflict graph of C' in G'' is a proper subgraph of the conflict graph of C in G'. In particular, the bridges containing u and v belong to different components of the conflict graph of C'. We may then assign each bridge to one of the two regions of the cycle C', in such a way that the bridges containing u and v are assigned consistently with the embedding $\mathcal{H}''$, and the remaining bridges are assigned in such a way that no two bridges in the same region conflict.

It is easy to see that any collection of C'-bridges that does not have a conflict can be embedded inside a single region of C' without crossing. Thus, $(G'', H'', \mathcal{H}'')$ is planar.

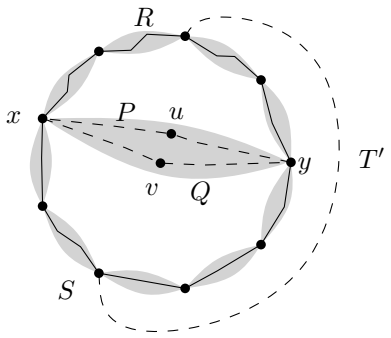


Figure 7.8.: Illustration of Lemma 7.16, the bridge embedded in the cycle contains a happy vertex u and an unhappy vertex v that are not connected by a path avoiding x and y. In this case, the PEG contains obstruction 4.

By analogous arguments, we see that removing or relaxing an edge or vertex of H' yields a planar PEG. Contracting an edge incident to u or v yields an planar PEG as well. Thus, $(G', H', \mathcal{H}')$ is a minimal obstruction. □

At least one of the embeddings is edge-compatible. Finally, we use all this preparation to analyze the skeletons of R-nodes. Suppose that $(G, H, \mathcal{H})$ is a 2-connected obstruction-free PEG, and that $\mathfrak{R}$ is an R-skeleton of G with at least one edge-compatible embedding. Let us assume that we have fixed an edge-compatible embedding $\mathfrak{R}^+$ for $\mathfrak{R}$.

Let C be a cycle of $\mathfrak{R}$ that is a projection of a cycle C' of $\mathcal{H}$. Recall that a vertex x of H that does not belong to an edge of C is happy with C' if it is embedded on the correct side of C in $\mathfrak{R}^+$ and that it is unhappy, otherwise. Recall further that a C-bridge is happy with C', if it contains a happy vertex, and it is unhappy if it contains an unhappy vertex and that a bridge that is neither happy nor unhappy is indifferent. We first show that a C-bridge cannot be happy and unhappy at the same time.

Lemma 7.16. *If C is a cycle of $\mathfrak{R}$ that is a projection of a cycle C' of $\mathcal{H}$, then no C-bridge can be both happy and unhappy with C'.*

Proof. Assume a C-bridge X contains a happy vertex u and an unhappy vertex v.

If there exists a G-path from u to v that avoids all the vertices of C', then we obtain obstruction 1. Assume then that there is no such path. This easily implies that the bridge X is a single $\mathfrak{R}$-edge B with two attachments x and y. Figure 7.8 shows this situation and illustrates the following steps. Since the graph represented by B is biconnected, there is a cycle D of G containing both u and v, and which is contained in B. Since every G-path from u to v inside B intersects x or y, we conclude that D can be expressed as a union of two G-paths P and Q from x to y, with $u \in P$ and $v \in Q$.

Similarly, the cycle C of $\mathfrak{R}$ can be expressed as a union of two $\mathfrak{R}$-paths R and S, each with at least one internal vertex. The paths R and S are projections of two H-paths R' and S'. Since $\mathfrak{R}$ is 3-connected, it has a path T that connects an internal vertex of R to an internal vertex of S, and whose internal vertices avoid C. The path T is a projection of a G-path T'. The paths P, Q, R', S' and T' can be contracted to form obstruction 4. □

Recall that a cycle C in $\mathfrak{R}$ that is a projection of a cycle C' of H, is happy, if there is at least one C-bridge that is happy with C' and it is unhappy, if at least one C-bridge is unhappy with C'. Again, as the following lemma shows, cycles cannot be both happy and unhappy at the same time.

Lemma 7.17. *If C' is a cycle of H whose projection is a cycle C of $\mathfrak{R}$, then C' cannot be both happy and unhappy.*

Proof. Suppose C has a happy bridge X containing a happy vertex u, and an unhappy bridge Y with an unhappy vertex v.

If C is a triangle, then X and Y cannot be chords of C and therefore they have three attachments, each. This implies that they are embedded on different sides of the triangle and all vertices of the triangle are attachments of both X and Y. Since Y is unhappy, it contains a vertex that is prescribed on the same side of C' as X. This yields obstruction 17. Otherwise, C has length at least 4, and we know that the reduced conflict graph K_C^- is connected by Lemma 7.11. We find a shortest path $X_1, \ldots, X_k$ in K_C^- connecting $X = X_1$ to $Y = X_k$. If the path is a single edge, we obtain obstruction 16. Otherwise we use Lemma 7.12 to choose for each X_i a pair of attachments $\{x_i, y_i\} \subseteq \mathrm{att}(X_i)$, such that $\{x_i, y_i\}$ alternates with $\{x_{i+1}, y_{i+1}\}$.

Since each C-bridge of the skeleton represents a connected subgraph of G, we know that for every $i = 2, \ldots, k-1$ the graph G has a path from x_i to y_i whose internal vertices avoid C' and which projects to the interior of X_i. We also know that there is a G-path Q_1 from x_1 to u, and a G-path R_1 from y_1 to u whose internal vertices avoid C' and which project into X_1. Similarly, there are G-paths Q_k and R_k from x_k to v and from y_k to v, internally disjoint with C' and projecting into X_k. Performing contractions if necessary, we may assume that all these paths are in fact single edges.

Consider the sub-Peg $(G', H', \mathcal{H}')$, where H' consists of the cycle C' and the two vertices u and v, and G has in addition all the edges obtained by contracting the paths defined above. If C' has shrinkable edges, we may contract them, until no shrinkable edges are left. Then we either obtain obstruction 4 (if $k = 3$), or Corollary 7.1 implies that have obtained an occurrence of Ach_k for some $k \geq 4$. □

Next, we would like to show that it is not possible that one cycle is happy and another one is unhappy. However, this is complicated if the cycles are too close in $\mathfrak{R}$, in particular if they share vertices. Therefore, we first show that an unhappy cycle C' projecting to a cycle C may not have an incident H-edge that does not belong to C. Such an edge e, if it existed, would either be a chord of C', or it would be part of a bridge containing a vertex of H (for example, the endpoint of e not belonging to C'). The next two lemmas exclude these two cases separately.

In the former case, where e in a chord of C' that hence projects to a chord of C, we also call e a *relevant chord*. Note that if B is an edge of $\mathfrak{R}$ containing a relevant chord, then in an edge-compatible embedding of $\mathfrak{R}$, B must always be embedded on the correct side of C. For practical purposes, such an edge B behaves as a happy bridge, as shown by the next lemma.

Lemma 7.18. *Let C' be a cycle of H that projects to a cycle C of $\mathfrak{R}$. Let e be a relevant chord of C' that projects into an $\mathfrak{R}$-edge B. Then C' cannot be unhappy.*

Proof. Let u and v be the two vertices of e, which are also the two poles of B. Let α' and β' be the two arcs of C' determined by the two vertices u and v, and let α and β be the two arcs of C that are projections of α' and β', respectively. Note that each of the two arcs α and β has at least one internal vertex, otherwise e would not be relevant.

Suppose for contradiction that C has an unhappy bridge X containing an unhappy vertex x. We distinguish two cases, depending on whether B is part of X or not.

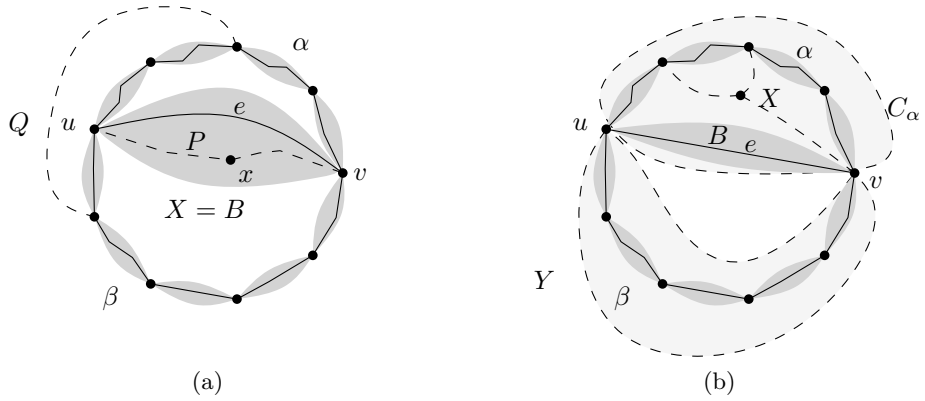


Figure 7.9.: Illustration of Lemma 7.18. The edge e is a relevant chord of the shown cycle, and the bridge X is assumed to be unhappy. If the skeleton edge of e also contains an unhappy vertex x, we obtain obstruction 3 (a). Otherwise, the skeleton edge of e together with the arc between its attachments forms a smaller cycle C_α, for which X is still unhappy, but the remainder of the cycle is part of a happy bridge (b), which contradicts the fact that a cycle cannot be both happy and unhappy.

First, assume that the bridge X contains the $\mathfrak{R}$-edge B. Then X is a trivial bridge whose only edge is B; see Figure 7.9a. The edge B then contains a G-path P from u to v containing x. The graph Gs also has a path Q connecting an internal vertex of α to an internal vertex of β and avoiding both u and v. Together, the edge e, the paths P and Q, and the arcs α and β can be contracted to form obstruction 3.

Assume now that the bridge X does not contain B. Consider two H-cycles $C'_\alpha = \alpha' \cup e$ and $C'_\beta = \beta' \cup e$, and their respective projections $C_\alpha = \alpha \cup B$ and $C_\beta = \beta \cup B$. It is not hard to see that the vertex x must be unhappy with at least one of the two cycles C'_α and C'_β. Let us say that X is unhappy with C'_α; see Figure 7.9b. Thus, C_α has at least one unhappy bridge. We claim that C_α also has a happy bridge. Indeed, let Y be the bridge of C_α that contains β. Since β has at least one internal vertex, the bridge Y is not indifferent. The bridge Y must be happy, otherwise the vertices u and v would violate edge-compatibility. This means that C_α has both a happy bridge and an unhappy bridge, contradicting Lemma 7.17. □

Lemma 7.19. *Let C' be a cycle of H that projects to a cycle C of $\mathfrak{R}$. If C' is unhappy, then every edge of H that is incident to a vertex of C must project into an $\mathfrak{R}$-edge that belongs to C.*

Proof. For contradiction, assume that an edge $e = uv$ of H is incident to a vertex $u \in C$, but projects into an $\mathfrak{R}$-edge $B \notin C$. If v is also a vertex of C, then e is a relevant chord and C may not have any unhappy bridges by Lemma 7.18. If $v \notin C$, then v is an internal vertex of a C-bridge, and from edge-compatibility it follows that v is happy with C'. Thus C has both happy and unhappy bridges, contradicting Lemma 7.17. □

The previous two lemmas show that for an unhappy cycle C' of H projecting to a cycle C of $\mathfrak{R}^+$, none the C-bridges contains an H-edge incident to a vertex of C. In particular, the projection of any happy H-cycle is either disjoint from C (that is they are far apart) or it is identical to C. We now exclude the latter case.

Lemma 7.20. *Let C'_1 and C'_2 be two distinct facial cycles of $\mathcal{H}$, which project to the same (undirected) cycle C of $\mathfrak{R}$. Then any C-bridge that is happy with C'_1 is also happy with C'_2.*

Proof. Let F_1 and F_2 be the faces of $\mathcal{H}$ corresponding to facial cycles C'_1 and C'_2, respectively.

Suppose for contradiction that at least one C-bridge X is happy with C'_1 and unhappy with C'_2. In view of Lemma 7.16, we may assume that X contains in its interior a vertex $x \in H$, such that x is happy with C'_1 and unhappy with C'_2.

Suppose that the two facial cycles C'_1 and C'_2 are oriented in such a way that their corresponding faces are to the left of the cycles. Note that any vertex of C is a common vertex of C'_1 and C'_2. This shows that the two facial cycles have at least three common vertices, which implies that they correspond to different faces of $\mathcal{H}$.

Let a, b and c be any three distinct vertices of C, and assume that these three vertices appear in the cyclic order (a, b, c) when the cycle C'_1 is traversed according to its orientation. The interior of the face F_2 lies to the right of the cycle C'_1, and in particular, the three vertices a, b, c appear in the cyclic order (c, b, a) when the boundary of F_2 is traversed in the orientation of C'_2. Thus, C'_1 and C'_2 induce opposite orientations of their common projection C. Since x is happy with exactly one of the two cycles C'_1 and C'_2, it means that in the graph H with embedding $\mathcal{H}$, the two cycles either both have x on their right, or both have x on their left. It is impossible that both facial cycles have x on their left, because they are facial cycles of different faces. Hence x is to the right of C'_1 and C'_2.

Let H_C be the connected component of H containing the vertices of C, and let $\mathcal{H}_C$ be its embedding inherited from $\mathcal{H}$. By Lemma 7.19, the bridge X contains no edge of H adjacent to C, so $x \notin H_C$. Let F_3 be the face of $\mathcal{H}_C$ that contains x in its interior. Note that F_3 is distinct from F_1 and F_2, as x is contained in it, which is not the case for F_1 and F_2. All the attachments of the bridge X must belong to the boundary of F_3 (as well as F_1 and F_2), otherwise we would obtain obstruction 1, using the fact that X contains a G-path from x to any attachment of X. If X has at least three attachments, this leads to contradiction, because no three faces of a planar graph can share three common boundary vertices — to see this, imagine inserting a new vertex into the interior of each of the three faces and connecting the new vertices by edges to the three common boundary vertices, to obtain a planar drawing of $K_{3,3}$.

Suppose now that X only has two attachments u and v, which means that X is a trivial bridge. Each of the two arcs of C determined by u and v must have an internal vertex. Let y and z be such internal vertices of the two arcs. To get a contradiction, insert a new vertex w into the interior of face F_1 in $\mathcal{H}$ and connect it by edges to all the four vertices u, v, y, z. Then draw an edge uv inside face F_3 and an edge yz inside F_2. The new edges together with the cycle C'_1 form a subdivision of K_5. □

We are now ready to show that $\mathfrak{R}^+$ may not have a happy and an unhappy cycle.

Lemma 7.21. *Let C'_1 and C'_2 be two cycles of H that project to two distinct cycles C_1 and C_2 of $\mathfrak{R}$. If C'_1 is unhappy, then C'_2 cannot be happy.*

Proof. Suppose that C'_1 is unhappy and C'_2 is happy. By Lemma 7.19, this means that no C_1-bridge may contain an edge of H incident to a vertex of C_1. Consequently, the two cycles C_1 and C_2 are vertex-disjoint. Since $\mathfrak{R}$ is 3-connected, it contains three disjoint paths P_1, P_2 and P_3, each connecting a vertex of C_1 to a vertex of C_2. Each path P_i is a projection of a G-path P'_i connecting a vertex of C'_1 to a vertex of C'_2. Note that C_1 is inside a happy bridge of C_2, and C_2 is inside an unhappy bridge of C_1. Thus, contracting the cycles C'_1 and C'_2 to triangles and contracting the paths P'_i to edges, we obtain obstruction 18. □

The next lemma shows that if any vertex u of $\mathfrak{R}$ that requires the embedding $\mathfrak{R}^+$, then no cycle can be unhappy.

Lemma 7.22. *Assume that H has three edges e_1, e_2 and e_3 which are incident to a common vertex u and project into three distinct $\mathfrak{R}$-edges B_1, B_2 and B_3 of $\mathfrak{R}$. Then no cycle of H that projects to a cycle of $\mathfrak{R}$ can be unhappy.*

Proof. Proceed by contradiction. Assume that there is an unhappy cycle C' of $\mathcal{H}$, which projects to a cycle C of $\mathfrak{R}$. From Lemma 7.19 it then follows that u does not belong to C, and hence u must belong to an unhappy C-bridge. From the same lemma we also conclude that the vertex u and the three edges e_i belong to a different component of H than the cycle C'.

For $i \in \{1, 2, 3\}$, suppose that the H-edge e_i connects vertex u to a vertex v_i, and is contained in an $\mathfrak{R}$-edge B_i that connects vertex u to a vertex w_i. These vertices, H-edges and $\mathfrak{R}$-edges are distinct, except for the possibility that $v_i = w_i$.

Since $\mathfrak{R}$ is 3-connected, all the vertices of $\mathfrak{R}$ that share a face with u belong to a common cycle D of $\mathfrak{R}$. The three vertices w_1, w_2 and w_3 split D into three internally disjoint arcs α_{12}, α_{13} and α_{23}, where α_{ij} has endvertices w_i and w_j.

As $\mathfrak{R}$ is 3-connected, it contains three disjoint paths P_1, P_2 and P_3, where P_i connects w_i to a vertex of C. We now distinguish two cases, depending on whether the paths P_i can avoid u or not.

First, assume that it is possible to choose the paths P_i in such a way that all of them avoid the vertex u. We may then add B_i to the path P_i to obtain three paths from u to C, which only share the vertex u. It follows that the graph G contains three paths R'_1, R'_2 and R'_3 from u to C' which are disjoint except for sharing the vertex u, and moreover, each R'_i contains the edge e_i. This yields obstruction 19.

Next, assume that it is not possible to choose P_1, P_2 and P_3 in such a way that all the three paths avoid u.

For $i \in \{1, 2, 3\}$, let x_i be the last vertex of P_i that belongs to D, assuming the path P_i is traversed from w_i towards C. Let Q_i be the subpath of P_i starting in x_i and ending in a vertex of C (so Q_i is obtained from P_i by removing vertices preceding x_i). Let y_1, y_2 and y_3 be the endvertices of P_1, P_2 and P_3 that belong to C. We may assume that y_i is the only vertex of P_i belonging to C, otherwise we could replace P_i with its proper subpath.

We claim that one of the three arcs α_{12}, α_{13}, and α_{13} must contain all the three vertices x_i, possibly as endvertices. If the vertices x_i did not belong to the same arc, we could connect each x_i to a unique vertex w_j by using the edges of D, and we would obtain three disjoint paths from w_i to C that avoid u. Assume then, without loss of generality, that α_{12} contains all the three vertices x_i.

We may also see that if the cycles C and D share a common vertex y, then y belongs to α_{12}. If not, we could connect w_3 to y by an arc of D that avoids w_1 and w_2, and we could connect w_1 and w_2 to two distinct vertices x_i and x_j by disjoint arcs of D, thus obtaining three disjoint paths from w_i to C avoiding u.

Fix distinct indices $p, q, r \in \{1, 2, 3\}$ so that the three vertices x_1, x_2 and x_3 are encountered in the order x_p, x_q, x_r when α_{12} is traversed in the direction from w_1 to w_2. Let β be the arc of D contained in α_{12} whose endpoints are x_p and x_r. Clearly x_q is an internal vertex of β.

We claim that at least one internal vertex of β is connected to u by an edge of $\mathfrak{R}$. Assume that this is not the case. Then we may insert into the embedding $\mathfrak{R}^+$ a new edge f connecting x_p and x_r and embedded inside the face of $\mathfrak{R}^+$ shared by x_q and u. Let γ be

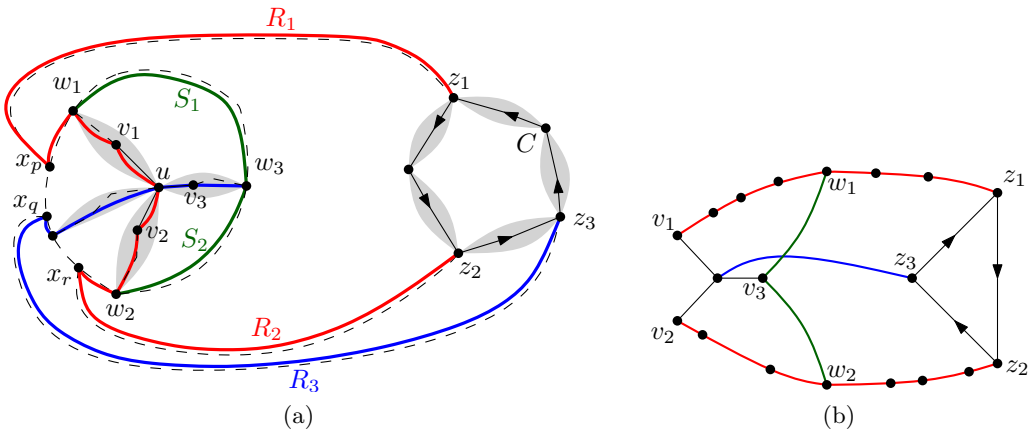


Figure 7.10.: Illustration of Lemma 7.22, the paths constructed in the proof (a) and an intermediate step in obtaining one of the obstructions 20, 21 or 22 (b).

the arc of C with endvertices y_p and y_r that does not contain y_q. The arc γ, the paths Q_p and Q_r and the edge f together form a cycle in the (multi)graph $\mathfrak{R} \cup \{f\}$. The vertex x_q and the vertex w_q are separated from each other by this cycle. Thus, the path P_q must share at least one vertex with this cycle, but that is impossible, since P_q is disjoint from Q_p, Q_r and γ. We conclude that $\mathfrak{R}$ has an $\mathfrak{R}$-edge B_4 connecting u to a vertex x_4 in the interior of β.

We define three paths R_1, R_2 and R_3 of the graph G as follows. The path R_1 starts in the vertex u, contains the edge $e_1 = uv_1$, proceeds from v_1 to w_1 inside B_1, then goes from w_1 to x_p inside the arc α_{12}, then follows Q_p until it reaches the vertex y_p. Similarly, the path R_2 starts in u, contains the edge e_2, follows from v_2 to w_2 inside B_2, from w_2 to x_r inside α_{12}, and then along Q_r to y_r. The path R_3 starts at the vertex w_3, proceeds towards v_3 inside B_3, then using the edge e_3 it reaches u, proceeds from u to x_4 inside B_4, then from x_4 to x_q inside β, then from x_q towards y_q along Q_q. If any of the three paths R_i contains more than one vertex of C', we truncate the path so that it stops when it reaches the first vertex of C'.

We also define two more paths S_1 and S_2 of G, where each S_i connects the vertex w_i to the vertex w_3 and projects into the arc α_{i3}; see Figure 7.10a for an illustration of the constructed paths.

Note that the three paths R_i only intersect in the vertex u, a path S_i may only intersect R_j at one of the vertices w_1, w_2 or w_3, and the cycle C' may intersect S_i only in the vertex w_i.

Consider the PEG $(G', H', \mathcal{H}')$ formed by the union of the cycle C', the three paths R_i, and the two paths S_j, where only the cycle C' and the three edges e_1, e_2 and e_3 with their vertices have prescribed embedding, and their embedding is inherited from $\mathcal{H}$.

It can be easily checked that the graph G' is a subdivision of a 3-connected graph, so it has a unique edge-compatible embedding $\mathcal{G}'$. Consider the subgraph $\mathfrak{R}'$ of $\mathfrak{R}$ formed by all the vertices of $\mathfrak{R}$ belonging to G' and all the $\mathfrak{R}$-edges that contain at least one edge of G'. The graph G' is a subdivision of $\mathfrak{R}'$. Thus, the embedding of $\mathfrak{R}'$ inherited from $\mathfrak{R}^+$ must have the same rotation schemes as the embedding $\mathcal{G}'$. Let z_i be the endpoint of R_i belonging to C'. Orient C' so that z_1, z_2, z_3 appear in this cyclic order on C'. Suppose that e_1, e_2 and e_3 appear in this clockwise order in $\mathcal{H}$. Then the four vertices u, v_1, v_2 and v_3 are to the left of C' in $\mathcal{G}'$, and hence also in $\mathfrak{R}^+$. Since the four vertices are in an unhappy C-bridge of $\mathfrak{R}$, they are to the right of C' in $\mathcal{H}'$. This determines $(G', H', \mathcal{H}')$ uniquely.

We now show that $(G', H', \mathcal{H}')$ contains one of the obstructions 20, 21 or 22. First, we contract each of S_1 and S_2 to a single edge. We also contract the cycle C' to a triangle with vertices z_1, z_2 and z_3. We contract the subpath of R_3 from w_3 to v_3 to a single vertex, and we contract the subpath of R_3 from u to z_3 to a single edge. After reversing the order of the vertices on the cycle to make it happy, we essentially obtain the PEG shown in Figure 7.10b, except that for $i = 1, 2$ it may be that $w_i = v_i$ or $w_i = z_i$, but not both since $v_i \neq z_i$. This is already very close to obstructions 20–22.

To contract R_1, we distinguish two cases. First, assume that w_1 belongs to C'. This means that $z_1 = w_1 \neq v_1$, because we know that v_1 is not in the same component of H as C'. In this case, we contract the subpath of R_1 from v_1 to w_1 to a single edge. On the other hand, if w_1 does not belong to C, we contract the subpath of R_1 from v_1 to w_1 to a single vertex, and we contract the subpath from w_1 to z_1 to a single edge.

The contraction of R_2 is analogous to the contraction of R_1, and it again depends on whether w_2 belongs to C or not. After these contractions are performed, we end up with one of the three obstructions 20, 21, or 22. □

With the lemmas proven so far, we are ready to prove the following proposition.

Proposition 7.4. *Let $(G, H, \mathcal{H})$ be an obstruction-free* PEG*, with G biconnected. Let $\mathfrak{R}$ be a skeleton of an R-node of the SPQR tree of G. If $\mathfrak{R}$ has at least one edge-compatible embedding, then it has a compatible embedding.*

Proof. Let $\mathfrak{R}^+$ be an edge-compatible embedding. If the embedding $\mathfrak{R}^+$ is not cycle-compatible, then $\mathcal{H}$ has an unhappy facial cycle C' projecting to a cycle C of $\mathfrak{R}$. The previous lemmas then imply that every facial cycle of $\mathcal{H}$ projecting to a cycle of $\mathfrak{R}$ can only have unhappy or indifferent bridges. Besides, Lemma 7.22 implies that no vertex u of $\mathfrak{R}$ can be incident to three $\mathfrak{R}$-edges, each of them containing an edge of H incident to u. Hence, the skeleton $\mathfrak{R}$ has no edge-compatibility constraints. Consequently, we may flip the embedding $\mathfrak{R}^+$ to obtain a new embedding that is compatible. □

This conclude our treatment of R-nodes and thus also the proof of the main theorem for biconnected PEGs. We now turn to 1-connected PEGs, that is, PEGs that are connected but not necessarily biconnected and to disconnected PEGs.

7.4. Disconnected and 1-Connected Pegs

We have shown that a biconnected obstruction-free PEG is planar. We now extend this characterization to arbitrary PEGs. To do this, we will first show that an obstruction-free PEG $(G, H, \mathcal{H})$ is planar if and only if each connected component of G induces a planar sub-PEG. Next, we provide a more technical argument showing that a connected obstruction-free PEG $(G, H, \mathcal{H})$ is planar, if and only if all the elements of a certain collection of 2-connected PEG-minors of $(G, H, \mathcal{H})$ are planar.

Reduction to G connected Here, we again make use of the combinatorial characterization we have established in Chapter 6. There we have proven the following lemma.

Lemma 7.23 (cf. Lemma 6.5 of Chapter 6)**.** *Let $(G, H, \mathcal{H})$ be a* PEG*. Let $G_1, \ldots, G_t$ be the connected components of G. Let H_i be the subgraph of H induced by the vertices of G_i, and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Then $(G, H, \mathcal{H})$ is planar if and only if 1) each*

$(G_i, H_i, \mathcal{H}_i)$ is planar, and 2) for each i, for each facial cycle $\vec{C}$ of H_i and for every $j \neq i$, no two vertices of H_j are separated by $\vec{C}$, in other words, all the vertices of H_j are embedded on the same side of C.

A PEG that does not satisfy the second condition of the lemma must contain obstruction 1. Thus, if Theorem 7.1 holds for PEGs with G connected, it holds for all PEGs.

Reduction to G biconnected Next, we consider connected PEGs $(G, H, \mathcal{H})$, that is PEGs where G is connected. In contrast to planarity of ordinary graphs, it is not in general true that a PEG is planar if and only if each sub-PEG induced by a biconnected component of G is planar. However, for PEGs satisfying some additional assumptions, a similar characterization is possible.

Let $(G, H, \mathcal{H})$ be a connected PEG and let v be a cutvertex of G. We say that v is H-separating if at least two connected components of $G - v$ contain vertices of H.

Let $(G, H, \mathcal{H})$ be a connected PEG that avoids obstruction 1. Let v be an H-separating cutvertex of G that does not belong to H. Let x and y be two vertices of H that belong to different connected components of $G - v$, chosen in such a way that there is a path in G connecting x to y whose internal vertices do not belong to H. The existence of such a path implies that x and y share a face F of $\mathcal{H}$, otherwise H would contain a cycle separating x from y, creating obstruction 1. The face F is unique, because x and y belong to distinct components of H. It follows that any planar embedding of G that extends $\mathcal{H}$ must embed the vertex v in the interior of the face F. We define $H' = H \cup v$ and let $\mathcal{H}'$ be the embedding of H' obtained from $\mathcal{H}$ by inserting the isolated vertex v into the interior of the face F. As shown above, any planar embedding of G that extends $\mathcal{H}$ also extends $\mathcal{H}'$. We say that $(G, H', \mathcal{H}')$ is obtained from $(G, H, \mathcal{H})$ by *fixing* the cutvertex v.

Let $(G, H^+, \mathcal{H}^+)$ be a PEG that is obtained from $(G, H, \mathcal{H})$ by fixing all the H-separating cut-vertices of G not belonging to H. Note that each H^+-separating cutvertex is also H-separating, and vice versa. A planar embedding of G that extends $\mathcal{H}$ also extends $\mathcal{H}^+$ and in particular, $(G, H, \mathcal{H})$ is planar if and only if $(G, H^+, \mathcal{H}^+)$ is planar. We now show that this operation cannot create a new obstruction in $(G, H^+, \mathcal{H}^+)$.

Lemma 7.24. *Let $(G, H, \mathcal{H})$ be a connected* PEG *that avoids obstruction 1, and let $(G, H^+, \mathcal{H}^+)$ be the* PEG *obtained by fixing all the H-separating cut-vertices of G. Then $(G, H, \mathcal{H})$ contains a minimal obstruction X if and only if $(G, H^+, \mathcal{H}^+)$ contains X.*

Moreover, given an occurrence of X in $(G, H^+, \mathcal{H}^+)$, an occurrence of X in $(G, H, \mathcal{H})$ can be found efficiently.

Proof. First, note that since $(G, H, \mathcal{H})$ is a PEG-minor of $(G, H^+, \mathcal{H}^+)$, it suffices to prove that if $(G, H^+, \mathcal{H}^+)$ contains an obstruction $X = (G_X, H_X, \mathcal{H}_X)$ then we can efficiently find the same obstruction in $(G, H, \mathcal{H})$. This clearly holds in the case when H_X does not contain isolated vertices, because then any sequence of deletions, contractions and relaxations that produces X inside $(G, H^+, \mathcal{H}^+)$ will also produce X inside $(G, H, \mathcal{H})$.

Suppose now that H_X contains isolated vertices. Assume first that G_X is 2-connected. Let $G_1, \dots G_t$ be the 2-connected blocks of G, let H_i be the subgraph of H induced by the vertices of G_i, let $\mathcal{H}_i$ be the embedding of H_i inherited from $\mathcal{H}$, and similarly for H_i^+ and $\mathcal{H}_i^+$. If $(G, H^+, \mathcal{H}^+)$ contains X, then for some i, $(G_i, H_i^+, \mathcal{H}_i^+)$ contains X as well (here we use the fact that each H^+-separating cutvertex of G belongs to H^+). However, each $(G_i, H_i^+, \mathcal{H}_i^+)$ is a PEG-minor of $(G, H, \mathcal{H})$ — this is because any vertex v of H_i^+ that is not a vertex of H_i is connected to a vertex of H by a path that internally avoids G_i. By contracting all such paths, we obtain a copy of $(G_i, H_i^+, \mathcal{H}_i^+)$ inside $(G, H, \mathcal{H})$. Since

$(G_i, H_i^+, \mathcal{H}_i^+)$ contains X, so does $(G, H, \mathcal{H})$. We can also easily see that an occurrence of X in $(G, H, \mathcal{H})$ can be efficiently obtained from its occurrence in $(G, H^+, \mathcal{H}^+)$.

It remains to deal with the case when X is not 2-connected and H_X contains an isolated vertex. This means that X is obstruction 1. By assumption, $(G, H, \mathcal{H})$ does not contain obstruction 1. Suppose for contradiction that $(G, H^+, \mathcal{H}^+)$ contains obstruction 1. This means that H^+ contains a cycle C and a pair of vertices v and w separated by this cycle, and that there exists a path P of G that connects v and w and has no vertex in common with C.

If v is not a vertex of H, then v is an H-separating cutvertex. Therefore, there are two vertices x and y of H in distinct components of $G - v$ that both share a face F with v and are connected to v by paths P_x and P_y of G which do not contain any other vertex of H. Since x and y are in distinct components of H, at least one of them, say x, does not belong to the cycle C. Since x shares a face with v, it must be on the same side of C as v. By the same reasoning, the vertex w either belongs to H or there is a vertex $z \in H$ that appears on the same side of C as w and is connected to w by a G-path P_z whose internal vertices do not belong to H. In any case, we find a pair of vertices of H that are separated by C and are connected by a G-path that avoids C. This shows that $(G, H, \mathcal{H})$ contains obstruction 1, which is a contradiction. □

Note that it is possible to determine whether a given PEG contains obstruction 1 efficiently.

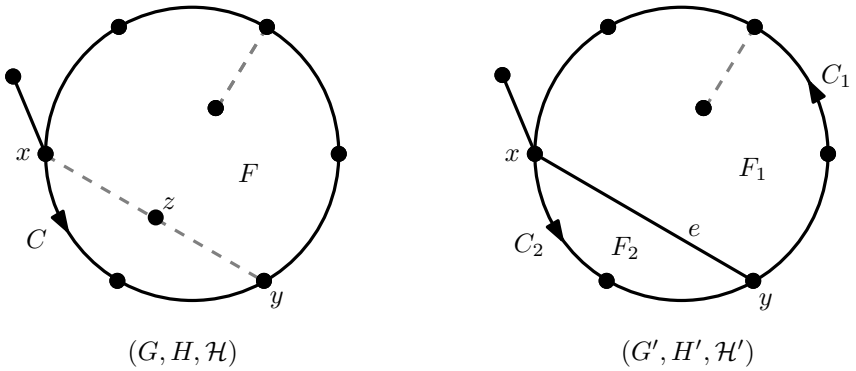
Lemma 7.24 shows that we can without loss of generality restrict ourselves to PEGs $(G, H, \mathcal{H})$ in which every H-separating cutvertex belongs to H. For PEGs having this property, we can show that planarity can be reduced to planarity of biconnected components.

First, we need a definition. Let H be a graph with planar embedding $\mathcal{H}$, let v be a vertex of H, and let H_1 and H_2 be two edge-disjoint subgraphs of H. We say that H_1 and H_2 *alternate* around v in $\mathcal{H}$, if there exist edges $e, e' \in E(H_1)$ and $f, f' \in E(H_2)$ which are all incident with v and appear in the cyclic order (e, f, e', f') in the rotation scheme of v in the embedding $\mathcal{H}$.

The following lemma is analogous to Lemma 6.4 from Chapter 6, except that the assumption "every non-trivial H-bridge is local" is replaced with the weaker condition "every H-separating cutvertex of G is in H". This new assumption is weaker, because a separating cutvertex not belonging to H necessarily belongs to a non-local H-bridge. However, the proof in Chapter 6 uses only this weaker assumption and therefore we have the following lemma.

Lemma 7.25. *Let $(G, H, \mathcal{H})$ be a connected* PEG *with the property that every H-separating cutvertex of G is in H. Let $G_1, \ldots, G_t$ be the blocks of G, let H_i be the subgraph of H induced by the vertices of G_i and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Then, $(G, H, \mathcal{H})$ is planar if and only if 1) $(G_i, H_i, \mathcal{H}_i)$ is a planar* PEG *for each i, 2) no two distinct graphs H_i and H_j alternate around any vertex of $\mathcal{H}$, and 3) for every facial cycle $\vec{C}$ of $\mathcal{H}$ and for any two vertices x and y of $\mathcal{H}$ separated by $\vec{C}$, any path in G connecting x and y contains a vertex of $\vec{C}$.*

The (already proven) main theorem for connected PEGs implies that the first condition holds for any obstruction-free PEG. Note that the last two conditions are always satisfied when $(G, H, \mathcal{H})$ avoids obstructions 1 and 2. Hence, this concludes the proof of Theorem 7.1. For the latter two conditions we can also efficiently test whether they are satisfied and produce an occurrence of an obstruction when one of them fails.



$(G, H, \mathcal{H})$ $(G', H', \mathcal{H}')$

Figure 7.11.: A reduction rule transforming $(G, H, \mathcal{H})$ into $(G', H', \mathcal{H}')$.

7.5. Other Minor-Like Operations

Let us remark that our definition of PEG-minor operations is not the only one possible. In this chapter, we preferred to work with a weaker notion of PEG-minors, since this makes the resulting characterization theorem stronger. However, in many circumstances, more general minor-like operations may be appropriate, providing a smaller set of minimal obstructions.

For example, the G-edge contraction rules may be relaxed to allow contractions in more general situations. Here is an example of such a relaxed G-edge contraction rule: given a PEG $(G, H, \mathcal{H})$, assume $e = uv$ is an edge of G but not of H, assume that u and v have a unique common face F of $\mathcal{H}$, and assume furthermore that each of the two vertices is visited only once by the corresponding facial walk of F. If u and v are in distinct components of H, or if the graph H is connected, we embed the edge uv into F and then contract it, resulting in a new PEG $(G', H', \mathcal{H}')$.

It is not hard to see that this relaxed contraction preserves the planarity of a PEG, and that $\mathcal{H}'$ is uniquely determined. It also subsumes the 'complicated G-edge contraction' we introduced. With this stronger contraction rule, most of the exceptional minimal obstructions can be further reduced, leaving only the obstructions 1, 2, 3, 4, 11, 14, 16, and 17, as well as K_5 and $K_{3,3}$. However, even this stronger contraction cannot reduce the obstructions from Ach_k.

To reduce the minimal obstructions to a finite set, we need an operation that can be applied to an alternating chain. We now present an example of such an operation. See Figure 7.11.

Suppose that $(G, H, \mathcal{H})$ is a PEG, let F be a face of $\mathcal{H}$, let C be a facial cycle of F oriented in such a way that the interior of F is to the left of C, let x and y be two vertices of C that are not connected by an edge of G, and let z be a vertex of H not belonging to C. Assume that the following conditions hold.

1. The vertex z is adjacent to x and to y in G.

2. The vertex z is embedded to the left of C in the embedding $\mathcal{H}$, and is incident to the face F.

3. Any connected component of H that is embedded to the left of C in $\mathcal{H}$ is connected to a vertex of $C \setminus \{x, y\}$ by an edge of G.

4. Any edge of H that is incident to x or to y and does not belong to C is embedded outside of F (that is, to the right of C) in $\mathcal{H}$.

We define a new PEG by the following steps.

- Remove vertex z and all its incident edges from G and H.
- Add to G, H and $\mathcal{H}$ a new edge $e = xy$. The edge e is embedded inside F. (Note that the position of e in the rotation schemes of x and y is thus determined uniquely, because of condition 4 above.)
- The edge e splits the face F into two subfaces F_1 and F_2. Let C_1 and C_2 be the facial cycles of F_1 and F_2 such that $C_1 \cup C_2 = C \cup \{e\}$. For any connected component B of H that is embedded to the left of C in H, let w be a vertex of $C \setminus \{x, y\}$ adjacent to a vertex of B. Such a vertex w exists by condition 3 above. If there are more such vertices, we choose one arbitrarily for each B. If w belongs to C_1, then B will be embedded inside F_1, otherwise it will be embedded inside F_2.

Let $(G', H', \mathcal{H}')$ be the resulting PEG. We easily see that if $(G, H, \mathcal{H})$ was planar, then $(G', H', \mathcal{H}')$ is planar as well. In fact, if the vertex z has degree 2 in G, then we may even say that $(G, H, \mathcal{H})$ is planar if and only if $(G', H', \mathcal{H}')$ is planar.

The operation described above allows to reduce each k-fold alternating chain with $k \geq 4$ to a smaller non-planar PEG which contains a $(k-1)$-fold alternating chain. It also reduces obstruction 4 to obstruction 3, and obstruction 16 to a PEG that contains obstruction 1. Therefore, when the above operation is added to the permissible minor operations, there will only be a finite number of minimal non-planar PEGs.

Let us point out that the obstructions from the infinite family $\bigcup_{k\geq 4} \mathrm{Ach}_k$ only play a role when cycle-compatibility is important. For certain types of PEGs, cycle-compatibility is not a concern. For instance, if the graph H is connected, it can be shown that $(G, H, \mathcal{H})$ is planar if and only if all the skeletons of G have edge-compatible embeddings, and therefore such a PEG is planar if and only if it avoids the finitely many exceptional obstructions.

7.6. Concluding Remarks

In this chapter we have shown that planar PEGs can be characterized by a finite set of forbidden obstructions, very similar to Kuratowski's characterization of planar graphs via the forbidden minors K_5 and $K_{3,3}$. The usual minor operations cannot directly be applied to PEGs. Therefore the results of this chapter lies outside the usual graph minor theory, and the powerful structural results of Robertson and Seymour [RS04] do not apply in this context. We therefore defined a set of PEG-minor operations and showed that with the right set of operations, the number of non-planar PEGs that are minimal with respect to these operations is finite.

It should be noted that Theorem 7.1 together with the linear-time algorithm for testing planarity of PEGs from Chapter 6 immediately implies Theorem 7.2, which states that a polynomial-time algorithm exists that, for a given PEG, either produces a planar embedding or an obstruction contained in it. In any non-planar instance $I = (G, H, \mathcal{H})$ only linearly many PEG-minor operations are possible. We test each one individually and use the linear-time testing algorithm to check whether the result is still non-planar. In this way we either find a smaller non-planar PEG I' resulting from I by one of the operations, or

we have found a minimal obstruction, which by Theorem 7.1 is contained in our list. The running time of this algorithm is at most $O(n^3)$.

In fact, in many cases, as indicated in several places, obstructions can be found much more efficiently, often in linear time. In particular, the linear-time testing algorithm gives an indication of which property of planar PEGs is violated for a given instance. In fact, almost all the proofs are constructive and can easily be implemented to run in linear time. The only exception is Proposition 7.2, which states that in a minimal wrung obstruction $(G, H, \mathcal{H})$ we have $V(G) = V(H)$. A linear-time algorithm for producing a wrung obstruction with this property from an arbitrary wrung obstruction would immediately imply a certifying PEG-planarity test with linear running time.

Open problems. Is it possible to find a minimal obstruction in a non-planar PEG in linear time? In general, given a fixed PEG $(G, H, \mathcal{H})$, what is the complexity of determining whether a given PEG contains $(G, H, \mathcal{H})$ as a PEG-minor? Here, the answer may depend on the set of PEG-minor operations we allow.

It is not known whether the results on planar PEGs can be generalized to graphs that have a partial embedding on a higher-genus surface. In fact, even the complexity of recognizing whether a graph partially embedded on a fixed higher-genus surface admits a crossing-free embedding extension is still an open problem.

Chapter 8

Simultaneous Embedding with Fixed Edges

In this chapter we study the time complexity of the problem *Simultaneous Embedding with Fixed Edges* (SEFE) that takes two planar graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ as input and asks whether a planar drawing Γ_1 of G_1 and a planar drawing Γ_2 of G_2 exist such that: (i) each vertex $v \in V$ is mapped to the same point in Γ_1 and in Γ_2; and (ii) every edge $e \in E_1 \cap E_2$ is mapped to the same Jordan curve in Γ_1 and Γ_2.

The SEFE problem was already mentioned in Chapter 6. The results there can be used to decide SEFE if the intersection of the two graphs has a fixed embedding. This happens, for example, when the intersection graph is triconnected. In this chapter, we study the SEFE problem in the more general cases when the intersection graph is connected or biconnected, but not necessarily triconnected.

First, we give a linear-time algorithm for SEFE when the *intersection graph* of G_1 and G_2, that is the planar graph $G_{1\cap 2} = (V, E_1 \cap E_2)$, is *biconnected*. Second, we show that SEFE, when $G_{1\cap 2}$ is *connected*, is equivalent to a suitably-defined *book embedding* problem. Based on this equivalence and on recent results by Hong and Nagamochi, we show a linear-time algorithm for the SEFE problem when $G_{1\cap 2}$ is a *star*. The chapter is based on joint work with Patrizio Angelini, Fabrizio Frati, Giuseppe Di Battista and Maurizio Patrignani [ADF+10b].

8.1. Introduction

Let $G_1 = (V, E_1), \ldots, G_k = (V, E_k)$ be k graphs on the same set of vertices. A *simultaneous embedding* of $G_1, \ldots, G_k$ consists of k planar drawings $\Gamma_1, \ldots, \Gamma_k$ of $G_1, \ldots, G_k$, respectively, such that any vertex $v \in V$ is mapped to the same point in every drawing Γ_i. Because of the applications to several visualization tasks and because of the interesting related theoretical problems, constructing simultaneous graph embeddings has recently grown up as a distinguished research topic in Graph Drawing.

The two main variants of the simultaneous embedding problem are the *geometric simultaneous embedding* and the *simultaneous embedding with fixed edges*. The former requires straight-line drawings of the input graphs, while the latter relaxes this constraint by just requiring the edges that are common to distinct graphs to be represented by the same Jordan curve in all the drawings.

Related work. Geometric simultaneous embedding turns out to have limited usability, as testing whether two planar graphs admit a geometric simultaneous embedding is NP-hard [EBGJ+07b] and as geometric simultaneous embeddings do not always exist if the input graphs are three paths [BCD+07], if they are two outerplanar graphs [BCD+07], if they are two trees [GKV09], and even if they are a tree and a path [AGKN10].

On the other hand, a simultaneous embedding with fixed edges (SEFE) always exists for much larger graph classes. Namely, a tree and a path always have a SEFE with few bends per edge [EK05], an outerplanar graph and a path or a cycle always have a SEFE with few bends per edge [DL07b], and a planar graph and a tree always have a SEFE [Fra06].

The main open question about the SEFE problem is whether testing the existence of a SEFE of two planar graphs is doable in polynomial time or not. A number of known results are related to this problem. Namely, Gassner et al. proved that testing whether three planar graphs admit a SEFE is NP-hard and that the SEFE problem is in $\mathcal{NP}$ for any number of input graphs [GJP+06]. Fowler et al. characterized the planar graphs that always have a SEFE with any other planar graph and have an efficient algorithm for testing whether two outerplanar graphs admit a SEFE [FJKS08]. Further, Fowler et al. showed how to test in polynomial time whether two planar graphs admit a SEFE if one of them contains at most one cycle [FGJ+08a]. Jünger and Schulz characterized the graphs $G_{1\cap 2}$ that allow for a SEFE of any two planar graphs G_1 and G_2 whose intersection graph is $G_{1\cap 2}$ [JS09]. Moreover, the results from Chapter 6 show how to test whether two planar graphs admit a SEFE if one of them has a fixed embedding.

Contribution and Outline. In this chapter, we show the following results. In Section 8.3 we present a linear-time algorithm for the SEFE problem when the intersection graph $G_{1\cap 2}$ of G_1 and G_2 is biconnected. Our algorithm exploits the SPQR-tree decomposition of $G_{1\cap 2}$ in order to test whether a planar embedding of $G_{1\cap 2}$ exists that allows the edges of G_1 and G_2 not in $G_{1\cap 2}$ to be drawn in such a way that no two edges of the same graph intersect. Haeupler et al. [HJL10] simultaneously and independently found a different linear-time algorithm for the same problem, based on PQ-trees.

In Section 8.4 we show that the SEFE problem, when $G_{1\cap 2}$ is connected, is equivalent to a suitably-defined book embedding problem. Namely, we show that for every instance (G_1, G_2) of SEFE such that $G_{1\cap 2}$ is connected, there exists a graph G', whose edges are partitioned into two sets E'_1 and E'_2, and a set of hierarchical constraints on the vertices of G', such that G_1 and G_2 have a SEFE if and only if G' admits a 2-page book embedding in which the edges of E'_1 are in one page, the edges of E'_2 are in another page, and the order of the vertices in V' along the spine respects the hierarchical constraints. Based on this characterization and on recent results by Hong and Nagamochi [HN09] concerning 2-page book embeddings with the edges assigned to the pages in the input, we prove that linear time suffices to solve the SEFE problem when $G_{1\cap 2}$ is a star.

8.2. Preliminaries

The graphs we consider in this chapter are connected, and we will consider embeddings in the plane, not on the sphere. Therefore embeddings in this chapter are described by the rotation scheme of the vertices and an outer face.

For a subgraph H of a graph G with planar embedding $\mathcal{E}$ we denote by $\mathcal{E}|_H$ the embedding of H induced by $\mathcal{E}$ and by ∂H the set of vertices of H that are adjacent to a

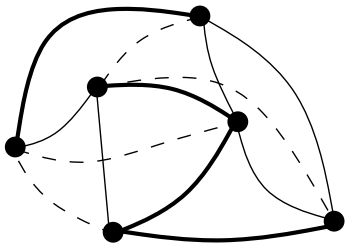


Figure 8.1.: A SEFE of two planar graphs. The edges that belong to both graphs are represented by solid fat segments, while the edges that belong to only one of the two graphs are represented by thin solid and dashed segments, respectively.

vertex of $G - H$. The following lemma is a very basic tool for manipulating embeddings.

Lemma 8.1 (Patching Lemma)**.** *Let $G = (V, E)$ be a biconnected planar graph with embedding $\mathcal{E}$ and let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two edge-disjoint biconnected subgraphs of G with $G_1 \cup G_2 = G$ and with the property that all the vertices of G_2 are in a single face f of $\mathcal{E}|_{G_1}$ (vertices in $V_1 \cap V_2$ are on the boundary of f). Further, let $\mathcal{E}'_2$ be an embedding of G_2 with the property that all the vertices of ∂G_2 are incident to the outer face of $\mathcal{E}'_2$ and appear in the same order as in $\mathcal{E}|_{G_2}$. Then there exists a planar embedding $\mathcal{E}'$ of G with $\mathcal{E}'|_{G_1} = \mathcal{E}|_{G_1}$ and $\mathcal{E}'|_{G_2} = \mathcal{E}'_2$.*

Proof. We prove the statement by induction on the number of vertices in $V_1 \cap V_2$.

For the base case, suppose that $V_1 \cap V_2 = \emptyset$. Let E' be the set of edges having one endvertex in V_1 and the other one in V_2. Remove from G all the edges of E' and change the embedding of G_2 to $\mathcal{E}'_2$. Since the order of the vertices along the outer face of $\mathcal{E}|_{G_2}$ and along the outer face of $\mathcal{E}'_2$ is the same, the edges in E' can be reinserted in a planar way, thus yielding an embedding $\mathcal{E}'$ of G with $\mathcal{E}'|_{G_1} = \mathcal{E}|_{G_1}$ and $\mathcal{E}'|_{G_2} = \mathcal{E}'_2$.

For the inductive case, suppose that $V_1 \cap V_2 \neq \emptyset$ and let $u \in V_1 \cap V_2$. By the assumption that all the vertices of G_2 are in a single face of $\mathcal{E}|_{G_1}$, the edges connecting u to the vertices of G_1 (respectively of G_2) form an interval in the cyclic ordering of the edges incident to u. We can therefore split u into two vertices u_1 and u_2 connected by the edge u_1u_2 such that u_i is connected to all the neighbors of u in G_i for $i = 1, 2$. Call G'' the resulting graph and modify G_1 and G_2 by renaming vertex u to u_i in G_i for $i = 1, 2$. Graphs G_1 and G_2 share one vertex less than before and hence, by induction, there exists an embedding $\mathcal{E}''$ of G'' with $\mathcal{E}''|_{G_1} = \mathcal{E}|_{G_1}$ and $\mathcal{E}''|_{G_2} = \mathcal{E}'_2$. We now undo the splitting operation by contracting the edge u_1u_2. This results in an embedding $\mathcal{E}'$ of G such that $\mathcal{E}'|_{G_1} = \mathcal{E}|_{G_1}$ and $\mathcal{E}'|_{G_2} = \mathcal{E}'_2$. □

Simultaneous embeddings. A *Simultaneous Embedding with Fixed Edges* (SEFE) of k planar graphs $G_1 = (V, E_1), G_2 = (V, E_2), \ldots, G_k = (V, E_k)$ consists of k drawings $\Gamma_1, \Gamma_2, \ldots, \Gamma_k$ such that: (i) Γ_i is a planar drawing of G_i, for $1 \leq i \leq k$; (ii) any vertex $v \in V$ is mapped to the same point in every drawing Γ_i, for $1 \leq i \leq k$; (iii) any edge $e \in E_i \cap E_j$ is mapped to the same Jordan curve in Γ_i and in Γ_j, for $1 \leq i, j \leq k$. The problem of testing whether k graphs admit a SEFE is called the SEFE problem. A SEFE of two planar graphs is depicted in Figure 8.1.

Given two planar graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, the *intersection graph* of G_1 and G_2 is the planar graph $G_{1\cap 2} = (V, E_1 \cap E_2)$; further, the *exclusive subgraph* of G_1 (resp. of G_2) is the graph $G_{1\setminus 2} = (V, E_1 \setminus E_2)$ (resp. $G_{2\setminus 1} = (V, E_2 \setminus E_1)$). The *exclusive*

edges of G_1 (of G_2) are the edges in $G_{1\setminus 2}$ (resp. in $G_{2\setminus 1}$). The *inclusive edges* of G_1 and G_2 are the edges in $G_{1\cap 2}$.

Jünger and Schulz [JS09] show that the Sefe problem can be equivalently stated in terms of embeddings. Namely, G_1 and G_2 admit a Sefe if and only if there exist planar embeddings $\mathcal{E}_1$ and $\mathcal{E}_2$ of G_1 and G_2, respectively, such that $\mathcal{E}_1|_{G_{1\cap 2}} = \mathcal{E}_2|_{G_{1\cap 2}}$ holds, that is, the two embeddings coincide when restricted to the intersection graph.

Book embeddings. A *book embedding* of a graph $G = (V, E)$ consists of a total ordering $\prec$ of the vertices in V and of an assignment of the edges in E to *pages* of a book, in such a way that no two edges ab and cd are assigned to the same page if $a \prec c \prec b \prec d$. A *$k$-page book embedding* is a book embedding using k pages. A *constrained k-page book embedding* is a k-page book embedding in which the assignment of edges to the pages is part of the input.

SPQR-tree. The main tool for handling the embeddings of the intersection graph will be the SPQR-tree. In this chapter, we will assume that any SPQR-tree of a graph G is rooted at one edge of G, called *reference edge*. Hence, the notion of the pertinent graph of a node and the parent of a node (except for the root node) are defined. We will show in Section 8.3 that the choice of this reference edge does not alter the possibility of constructing a Sefe.

In the following, we will only refer to the SPQR-tree of the intersection graph $G_{1\cap 2}$ of two graphs G_1 and G_2. For a node μ we denote its pertinent graph by $G_{1\cap 2}(\mu)$, and we denote by $G_1(\mu)$ (by $G_2(\mu)$) the subgraph of G_1 (of G_2) induced by the vertices in $G_{1\cap 2}(\mu)$ and by $G(\mu)$ the graph $G_1(\mu) \cup G_2(\mu)$. Note that the intersection graph $G_{1\cap 2}$ of two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, and its SPQR-tree $\mathcal{T}$ can be computed in linear time [GM00]. We say that a vertex v of $G_{1\cap 2}$ *belongs* to a node μ, if it is a vertex of $G_{1\cap 2}(\mu)$. In this case we also say that μ contains v.

8.3. Computing a Sefe When the Intersection Graph is Biconnected

In this section we show an algorithm for deciding whether a Sefe of two planar graphs G_1 and G_2 whose intersection graph $G_{1\cap 2}$ is biconnected exists. The description of the algorithm consists of two parts. In Section 8.3.1 we give combinatorial results characterizing the embeddings of $G_{1\cap 2}$ that allow for a Sefe in terms of conditions on the embeddings of the skeletons of its SPQR-tree. This yields a simple polynomial-time algorithm for testing the existence of a Sefe of G_1 and G_2. In Section 8.3.2 we use dynamic programming to improve the algorithm's running time to linear.

8.3.1. A Polynomial-Time Algorithm

Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two planar graphs whose intersection graph $G_{1\cap 2}$ is biconnected. Denote by $\mathcal{T}$ the SPQR-tree of $G_{1\cap 2}$.

To ease the description of the algorithm, we assume that $\mathcal{T}$ is rooted at any edge e of $G_{1\cap 2}$. This implies that e is adjacent to the outer face of any computed embedding of $G_{1\cap 2}$. Observe that this does not preclude the possibility of finding a Sefe of G_1 and G_2. Namely, consider any Sefe in the plane; "wrap" the Sefe around a sphere; project the

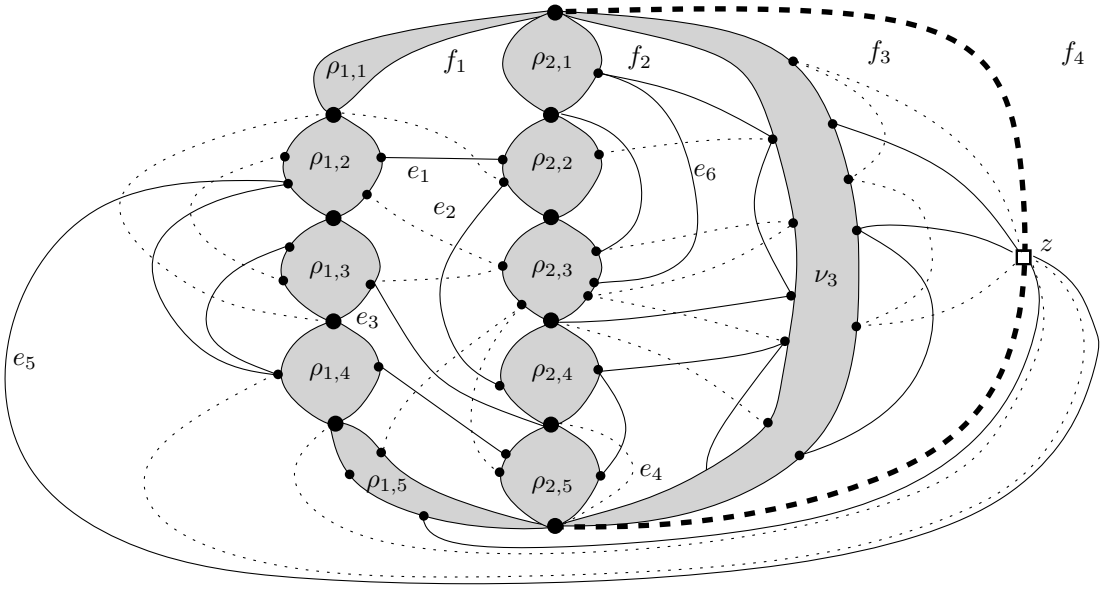


Figure 8.2.: A SEFE of graphs $G_1(\mu)$ and $G_2(\mu)$ when μ is a P-node with three children ν_1, ν_2, and ν_3. Also, ν_1 and ν_2 have children $\rho_{1,1}, \ldots, \rho_{1,5}$ and $\rho_{2,1}, \ldots, \rho_{2,5}$, respectively. For each visible node τ of μ, the interior of the cycle delimiting the outer face of $G_{1\cap 2}(\tau)$ is gray. Solid (dotted) edges are exclusive edges of G_1 (G_2). The dashed edge represents the rest of the graph.

SEFE back to the plane from a point in a face incident to e, thus obtaining a SEFE of G_1 and G_2 in which e is incident to the outer face of the embedding of $G_{1\cap 2}$. Further, if e is adjacent in $\mathcal{T}$ to an S-node, subdivide the edge of $\mathcal{T}$ connecting e to its only child by inserting a P-node. Observe that the described insertion of an artificial P-node ensures that the parent of any S-node is either an R-node or a P-node.

An exclusive edge e of G_1 or of G_2 is an *internal edge* of a node $\mu \in \mathcal{T}$ if both endpoints of e belong to μ, at least one of them is not a pole of μ, and there exists no descendant of μ containing both the endpoints of e. An exclusive edge e of G_1 or of G_2 is an *outer edge* of a node $\mu \in \mathcal{T}$ if exactly one endpoint of e belongs to μ and this endpoint is not a pole of μ. An exclusive edge e of G_1 or of G_2 is an *intra-pole edge* of a node $\mu \in \mathcal{T}$ if its endpoints are the poles of μ. Observe that an exclusive edge e of G_1 or of G_2 can be an outer edge of a linear number of nodes of $\mathcal{T}$; also, e is an internal edge of at most one node of $\mathcal{T}$; moreover, e can be an intra-pole edge of a linear number of nodes of $\mathcal{T}$; however, e can be an intra-pole edge of at most one P-node of $\mathcal{T}$. In Figure 8.2, edge e_1 is an internal edge of μ and an outer edge of $\rho_{1,2}$, of $\rho_{2,2}$, of ν_1, and of ν_2; edge e_2 is an internal edge of ν_2 and an outer edge of $\rho_{2,2}$ and $\rho_{2,4}$; edge e_3 is an internal edge of μ and an outer edge of $\rho_{1,3}$, ν_1, and ν_2; edge e_4 is an intra-pole edge of $\rho_{2,5}$; edge e_5 is an outer edge of $\rho_{1,2}$, of ν_1, and of μ.

The algorithm performs a bottom-up traversal of $\mathcal{T}$. When it visits a node μ of $\mathcal{T}$, either it concludes that a SEFE of G_1 and G_2 does not exist, or it determines a SEFE $\Gamma(\mu)$ of $G_1(\mu)$ and $G_2(\mu)$ such that, if a SEFE of G_1 and G_2 exists, there exists one in which the SEFE of the subgraphs $G_1(\mu)$ and $G_2(\mu)$ is $\Gamma(\mu)$. The rest of the graph, that is, the union of the graphs obtained from G_1 and G_2 by respectively removing the vertices of $G_1(\mu)$ and $G_2(\mu)$, except for the poles $u(\mu)$ and $v(\mu)$ of μ, and their incident edges, will be placed in the same connected region of $\Gamma(\mu)$. This region is called the *outer face* of $\Gamma(\mu)$. The

computed SEFE $\Gamma(\mu)$ of $G_1(\mu)$ and $G_2(\mu)$ has the property that all the outer edges of μ can be *drawn towards the outer face*, that is, a vertex z can be inserted into the outer face of $\Gamma(\mu)$ and all the outer edges of μ can be drawn with z replacing their endpoints not in μ, still maintaining the planarity of the drawings of $G_1(\mu)$ and $G_2(\mu)$. An example of the insertion of z in a SEFE of $G_1(\mu)$ and $G_2(\mu)$ is shown in Figure 8.2. Note that if we have found such a SEFE $\Gamma(\mu)$ of $G_1(\mu)$ and $G_2(\mu)$, then also its flip (obtained by mirroring the drawing) forms such a SEFE. To use this degree of freedom, our algorithm fixes the embedding of the pertinent graph of a processed node only up to a flip. The choice of the flip is then deferred until later, when processing the parent node (or even the parent of the parent) since this decision requires more global information that is available only later.

The algorithm does not process any S-node directly. First, an S-node does not offer embedding choices, and second, it also does not deliver enough information to decide on the flips of its children. Therefore choices of the flips of the children of a node μ are deferred to the step in which the parent of μ is processed. Then, for every P-node and every R-node μ of $\mathcal{T}$, the *visible nodes* of μ are the children of μ that are not S-nodes plus the children of each child of μ that is an S-node. Intuitively, the visible nodes of a P-node or R-node μ are the descendants ν of μ such that, when μ is processed, an embedding of $G(\nu)$ has been already decided up to a flip. In fact, when processing an R-node μ a flip for the embedding of the pertinent graph $G(\nu)$ of each visible node ν is determined and when processing a P-node μ an ordering of the children of μ around its poles and a flip for the embedding of the pertinent graph $G(\nu)$ of each visible node ν are determined.

Some of the embedding choices that are taken when processing a P-node or an R-node μ are forced by the existence of exclusive edges connecting different visible nodes of μ, as will be stated in Lemmas 8.2 and 8.3. Some other embedding choices are arbitrary. However, such arbitrary choices do not alter the possibility of finding a SEFE of G_1 and G_2. Namely, we will prove that for any P-node or R-node μ any SEFE $\Gamma(\mu)$ of $G_1(\mu)$ and $G_2(\mu)$ can be extended to a SEFE of G_1 and G_2 if the latter SEFE exists, provided that $\Gamma(\mu)$ allows for drawing the outer edges of μ towards the outer face without creating crossings. Thus, when processing a P-node or an R-node μ there is no need for looking at the rest of the graph in order to decide an embedding of $G(\mu)$ such that the possibility of finding a SEFE of G_1 and G_2 is not precluded. We start with two necessary conditions on the embeddings of the skeletons of the nodes of $\mathcal{T}$.

Lemma 8.2. *Let $\mathcal{E}_{1\cap2}(\mu)$ be an embedding of $G_{1\cap2}(\mu)$, with $\mu \in \mathcal{T}$, and let e be an internal edge of μ. Then, G_1 and G_2 have a* SEFE *in which the embedding of $G_{1\cap2}(\mu)$ is $\mathcal{E}_{1\cap2}(\mu)$ only if both endpoints of e are incident to the same face of $\mathcal{E}_{1\cap2}(\mu)$.*

Proof. The statement directly follows from the observation that, in any embedding $\mathcal{E}_{1\cap2}(\mu)$ of $G_{1\cap2}(\mu)$ in which the endpoints of e are not both incident to the same face, the edge e crosses at least one edge of $G_{1\cap2}(\mu)$. As the edges of $G_{1\cap2}(\mu)$ belong to both G_1 and G_2, either two edges of G_1 or two edges of G_2 cross (depending on whether $e \in G_1$ or $e \in G_2$). □

Lemma 8.3. *Let $\mathcal{E}_{1\cap2}(\mu)$ be an embedding of $G_{1\cap2}(\mu)$, with $\mu \in \mathcal{T}$, and let e be an outer edge incident to μ in a vertex $u(e)$. Then, G_1 and G_2 have a* SEFE *in which the embedding of $G_{1\cap2}(\mu)$ is $\mathcal{E}_{1\cap2}(\mu)$ only if $u(e)$ is on the outer face of $\mathcal{E}_{1\cap2}(\mu)$.*

Proof. Similar to Lemma 8.2, the statement follows from the observation that, in any embedding $\mathcal{E}_{1\cap2}(\mu)$ of $G_{1\cap2}(\mu)$ in which $u(e)$ is not incident to the outer face, edge e crosses at least one edge of $G_{1\cap2}(\mu)$. As the edges of $G_{1\cap2}(\mu)$ belong to both G_1 and G_2, either two edges of G_1 or two edges of G_2 cross (depending on whether $e \in G_1$ or $e \in G_2$). □

We now prove (in Lemma 8.4) that, in any SEFE Γ of G_1 and G_2 and for any node $\mu \in \mathcal{T}$ that is not an S-node, the outer face of $G(\mu)$ is (almost) the same. This will allow us to prove (in Lemma 8.5) that, if a SEFE of G_1 and G_2 exists, then (almost) *any* SEFE of $G_1(\mu)$ and $G_2(\mu)$ can be extended to a SEFE of G_1 and G_2.

Consider a SEFE $\Gamma = (\mathcal{E}_1, \mathcal{E}_2)$ of G_1 and G_2, and, for a node μ of $\mathcal{T}$, the outer face of $G_{1\cap 2}(\mu)$ in Γ. This face is delimited by a clockwise cycle C containing $u(\mu)$ and $v(\mu)$. Denote by $C_1(\Gamma, \mu)$ the circular list containing $u(\mu)$, $v(\mu)$, and all the vertices that are incident to exclusive edges of G_1 that are outer edges of μ, in the same order as they appear in C. Intuitively, $C_1(\Gamma, \mu)$ consists of the vertices of $\partial G_1(\mu)$ in their clockwise order of appearance along the outer face of $G_{1\cap 2}(\mu)$. List $C_2(\Gamma, \mu)$ is defined analogously, with G_1 replaced by G_2. We claim that, in each SEFE of G_1 and G_2 and for any node $\mu \in \mathcal{T}$ that is not an S-node, the lists $C_1(\Gamma, \mu)$ and $C_2(\Gamma, \mu)$ are essentially the same. Denote by C^{rev} the reverse of a circular list C.

Lemma 8.4. *For any two* SEFE Γ *and* Γ' *of* G_1 *and* G_2 *and for any node* $\mu \in \mathcal{T}$ *that is not an S-node, either* $C_1(\Gamma, \mu) = C_1(\Gamma', \mu)$ *and* $C_2(\Gamma, \mu) = C_2(\Gamma', \mu)$*, or* $C_1(\Gamma, \mu) = C_1^{\text{rev}}(\Gamma', \mu)$ *and* $C_2(\Gamma, \mu) = C_2^{\text{rev}}(\Gamma', \mu)$ *hold.*

Proof. Suppose, for a contradiction, that there exists two SEFEs Γ and Γ' and a node μ of $\mathcal{T}$ that is not an S-node for which the statement does not hold. We show that this implies that Γ or Γ' is actually not a SEFE of G_1 and G_2. Suppose, without loss of generality, that the statement holds for all the descendants of μ in $\mathcal{T}$.

If μ is a Q-node then $C_1(\Gamma, \mu) = C_1(\Gamma', \mu) = C_2(\Gamma, \mu) = C_2(\Gamma', \mu) = [u(\mu), v(\mu)]$ and the statement holds, thus obtaining a contradiction.

Suppose that μ is an R-node. Since the statement holds for every visible node of μ and since skel(μ) has exactly one planar embedding, up to a reversal of the adjacency lists of all the vertices, there exists a visible node of μ that is flipped differently in Γ and Γ' and that has an outer edge e that is also an outer edge of μ. Denote by $u(e)$ the endpoint of e belonging to μ. Suppose that $u(e)$ is incident to the outer face of $G_{1\cap 2}(\mu)$ in Γ. Then, $u(e)$ is not incident to the outer face of $G_{1\cap 2}(\mu)$ in Γ'. It follows that the edge e crosses $G_{1\cap 2}(\mu)$ in Γ', a contradiction.

Suppose that μ is a P-node. Then at most two children ν_x and ν_y of μ contain vertices of $\partial G_{1\cap 2}$ different from $u(\mu)$ and from $v(\mu)$, as otherwise a vertex of $\partial G_{1\cap 2}$ would not be incident to the outer face of $G_{1\cap 2}(\mu)$ in Γ and in Γ' and any outer edge of μ incident to such a vertex would cross $G_{1\cap 2}(\mu)$, thus contradicting the assumption that Γ and Γ' are SEFEs of G_1 and G_2. The flips of ν_x and ν_y in Γ (if ν_x and ν_y are not S-nodes) or the flips of the children of ν_x and ν_y in Γ (if ν_x and ν_y are S-nodes) determine circular lists $C_1(\Gamma, \mu)$ and $C_2(\Gamma, \mu)$. An analogous statement holds with Γ' replacing Γ. Then, analogously to the R-node case, if a visible node of μ has an outer edge e that is also an outer edge of μ and such a node is flipped differently in Γ and Γ', then the endpoint $u(e)$ of e in μ is not incident to the outer face of $G_{1\cap 2}(\mu)$ either in Γ or in Γ'. It follows that the edge e crosses $G_{1\cap 2}(\mu)$ in Γ or in Γ', a contradiction. □

Lemma 8.4 proves that the choice of an embedding $\mathcal{E}_{1\cap 2}(\mu)$ for $G_{1\cap 2}(\mu)$ does not restrict the possibility of finding a SEFE of G_1 and G_2 as long as the vertices in $\partial G_1(\mu)$ and $\partial G_2(\mu)$ are incident to the outer face of the computed SEFE of $G_{1\cap 2}(\mu)$. However, the condition that the vertices of $\partial G_1(\mu)$ and $\partial G_2(\mu)$ are incident to the outer face of $\mathcal{E}_{1\cap 2}(\mu)$ is not sufficient to guarantee that a SEFE of $G_1(\mu)$ and $G_2(\mu)$ can be extended to a SEFE of G_1 and G_2, if a SEFE of G_1 and G_2 exists. Namely, it is also necessary that all the vertices of $\partial G_i(\mu)$ are incident to the outer face of $G_i(\mu)$, for $i = 1, 2$, as otherwise the outer edges of

μ could not be drawn towards the outer face. A SEFE of $G_1(\mu)$ and $G_2(\mu)$ such that all the vertices of $\partial G_i(\mu)$ are incident to the outer face of $G_i(\mu)$, for $i = 1, 2$, is called *extendable*. We now show that any extendable SEFE of $G_1(\mu)$ and $G_2(\mu)$ can be extended to a full SEFE of G_1 and G_2, provided that a SEFE of G_1 and G_2 exists. In fact, we show a more general result, which is as follows. Denote by $G_1 \setminus G_1(\mu)$ (by $G_2 \setminus G_2(\mu)$) the subgraph obtained from G_1 (resp. from G_2) by removing all the edges in $G_1(\mu)$ (resp. in $G_2(\mu)$) and all the vertices in $G_1(\mu)$ (resp. in $G_2(\mu)$), except for the poles $u(\mu)$ and $v(\mu)$ of μ.

Lemma 8.5 (Simultaneous Patching Lemma)**.** *Let G_1 and G_2 be two planar graphs such that $G_{1\cap 2}$ is biconnected, let $\mathcal{T}$ be the SPQR-tree of $G_{1\cap 2}$, and let μ be a node of $\mathcal{T}$ that is not an S-node. Let $(\mathcal{E}_1^\mu, \mathcal{E}_2^\mu)$ be an extendable* SEFE *of $G_1(\mu)$ and $G_2(\mu)$ and let $(\mathcal{E}_1, \mathcal{E}_2)$ be a* SEFE *of G_1 and G_2. Then there exists a* SEFE *$(\mathcal{E}_1', \mathcal{E}_2')$ of G_1 and G_2 such that:*

1. *$\mathcal{E}_1'|_{G_1\setminus G_1(\mu)}$ and $\mathcal{E}_2'|_{G_2\setminus G_2(\mu)}$ coincide with $\mathcal{E}_1|_{G_1\setminus G_1(\mu)}$ and $\mathcal{E}_2|_{G_2\setminus G_2(\mu)}$, and*
2. *$\mathcal{E}_1'|_{G_1(\mu)}$ and $\mathcal{E}_2'|_{G_2(\mu)}$ coincide either with $\mathcal{E}_1^\mu$ and $\mathcal{E}_2^\mu$ or with their flips.*

Proof. First, $G_1(\mu)$ and $G_1 \setminus G_1(\mu)$ are edge-disjoint subgraphs of G_1 by construction. Second, the union of the vertex sets of $G_1(\mu)$ and $G_1 \setminus G_1(\mu)$ is the vertex set of G_1, by construction. Third, all the vertices of $G_1(\mu)$ are in a single face of $\mathcal{E}_1|_{G_1\setminus G_1(\mu)}$, with the common vertices (that is, $u(\mu)$ and $v(\mu)$) on the boundary of such a face. Fourth, all the vertices of $\partial G_1(\mu)$ are on the outer face of $\mathcal{E}_1^\mu$ since $(\mathcal{E}_1^\mu, \mathcal{E}_2^\mu)$ is an extendable SEFE of $G_1(\mu)$ and $G_2(\mu)$. Finally, the vertices of $\partial G_1(\mu)$ appear on the outer face of $\mathcal{E}_1^\mu$ with the same or with the reverse order as in $\mathcal{E}_1|_{G_1(\mu)}$, by Lemma 8.4.

Clearly, the same applies for $G_2(\mu)$ and $G_2 \setminus G_2(\mu)$ with embedding $\mathcal{E}_2$. Hence, after possibly flipping $\mathcal{E}_1^\mu$ and $\mathcal{E}_2^\mu$, the vertices of $\partial G_1(\mu)$ (of $\partial G_2(\mu)$) appear on the outer face of $\mathcal{E}_1^\mu$ (respectively of $\mathcal{E}_2^\mu$) with the same order as in $\mathcal{E}_1|_{G_1(\mu)}$ (respectively as in $\mathcal{E}_2|_{G_2(\mu)}$).

Therefore, Lemma 8.1 applies to G_1 (to G_2), that is, an embedding $\mathcal{E}_1'$ (respectively $\mathcal{E}_2'$) of G_1 (respectively of G_2) exists such that $\mathcal{E}_1'|_{G_1\setminus G_1(\mu)} = \mathcal{E}_1|_{G_1\setminus G_1(\mu)}$ and $\mathcal{E}_1'|_{G_1(\mu)} = \mathcal{E}_1^\mu$ (respectively $\mathcal{E}_2'|_{G_2\setminus G_2(\mu)} = \mathcal{E}_2|_{G_2\setminus G_2(\mu)}$ and $\mathcal{E}_2'|_{G_2(\mu)} = \mathcal{E}_2^\mu$). Since $(\mathcal{E}_1'|_{G_1\setminus G_1(\mu)}, \mathcal{E}_2'|_{G_2\setminus G_2(\mu)})$ is a SEFE of $(G_1 \setminus G_1(\mu), G_2 \setminus G_2(\mu))$ (given that $(\mathcal{E}_1, \mathcal{E}_2)$ is a SEFE of G_1 and G_2) and since $(\mathcal{E}_1'|_{G_1(\mu)}, \mathcal{E}_2'|_{G_2(\mu)})$ is a SEFE of $(G_1(\mu), G_2(\mu))$ (given that $(\mathcal{E}_1^\mu, \mathcal{E}_2^\mu)$ is a SEFE of $G_1(\mu)$ and $G_2(\mu)$), then $(\mathcal{E}_1', \mathcal{E}_2')$ is a SEFE of G_1 and G_2 with the required properties. □

As a corollary we that any extendable SEFE of the subgraphs induced by a non-S-nide can be extended to a complete SEFE of the two graphs, provided that one exists.

Corollary 8.1. *Let G_1 and G_2 be two planar graphs such that $G_{1\cap 2}$ is biconnected, let $\mathcal{T}$ be the SPQR-tree of $G_{1\cap 2}$, and let μ be a node of $\mathcal{T}$ that is not an S-node. If G_1 and G_2 admit a* SEFE *then any extendable* SEFE *of $G_1(\mu)$ and $G_2(\mu)$ or its flip can be extended to a* SEFE *of G_1 and G_2.*

Corollary 8.1 implies that, for a non-S-node $\mu \in \mathcal{T}$ with visible nodes $\mu_1, \ldots, \mu_k$, we can fix up to a flip extendable SEFEs $(\mathcal{E}(G_1(\mu_1)), \mathcal{E}(G_2(\mu_1))), \ldots, (\mathcal{E}(G_1(\mu_k)), \mathcal{E}(G_2(\mu_k)))$ of $(G_1(\mu_1), G_2(\mu_1)), \ldots, (G_1(\mu_k), G_2(\mu_k))$ without altering the possibility of finding a SEFE of $(G_1(\mu), G_2(\mu))$. Therefore, when processing μ we assume that the visible nodes $\mu_1, \ldots, \mu_k$ have fixed extendable SEFEs $(\mathcal{E}(G_1(\mu_1)), \mathcal{E}(G_2(\mu_1))), \ldots, (\mathcal{E}(G_1(\mu_k)), \mathcal{E}(G_2(\mu_k)))$ and we want to test whether an extendable SEFE of $(G_1(\mu), G_2(\mu))$ exists. Observe that the computation of an extendable SEFE of μ implies choosing an embedding of skel(μ) and a flip for the SEFEs $(\mathcal{E}(G_1(\mu_1)), \mathcal{E}(G_2(\mu_1))), \ldots, (\mathcal{E}(G_1(\mu_k)), \mathcal{E}(G_2(\mu_k)))$. Lemmas 8.2 and 8.3 give necessary conditions that the embedding of skel(μ) has to satisfy to lead to an

extendable SEFE of $(G_1(\mu), G_2(\mu))$. We call *compatible* an embedding of $\text{skel}(\mu)$ satisfying these conditions. We now show that given a compatible embedding $\mathcal{E}(\text{skel}(\mu))$ of $\text{skel}(\mu)$, if $G_1(\mu)$ and $G_2(\mu)$ admit an extendable SEFE, then they admit an extendable SEFE in which the embedding of $\text{skel}(\mu)$ is $\mathcal{E}(\text{skel}(\mu))$.

Theorem 8.1. *Let G_1 and G_2 be two planar graphs whose intersection graph $G_{1\cap 2}$ is biconnected and let $\mathcal{T}$ be the SPQR-tree of $G_{1\cap 2}$. Let μ be any node of $\mathcal{T}$ that is not an S-node and let $\mu_1, \ldots, \mu_k$ be the visible nodes of μ. Assume that an extendable* SEFE *$(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ of $(G_1(\mu_i), G_2(\mu_i))$ exists, for each $i = 1, \ldots, k$, and assume that a compatible embedding $\mathcal{E}(\text{skel}(\mu))$ of $\text{skel}(\mu)$ exists. Then, if $G_1(\mu)$ and $G_2(\mu)$ admit an extendable* SEFE*, they admit an extendable* SEFE *in which the embedding of $\text{skel}(\mu)$ is $\mathcal{E}(\text{skel}(\mu))$ and the* SEFE *of $(G_1(\mu_i), G_2(\mu_i))$ is either $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ or its flip, for each $i = 1, \ldots, k$.*

Proof. If $(G_1(\mu), G_2(\mu))$ do not admit an extendable SEFE then there is nothing to prove. Hence, assume that $(G_1(\mu), G_2(\mu))$ admit an extendable SEFE $(\mathcal{E}'(G_1(\mu)), \mathcal{E}'(G_2(\mu)))$. Let $\mathcal{E}'(\text{skel}(\mu))$ be the embedding of $\text{skel}(\mu)$ in $(\mathcal{E}'(G_1(\mu)), \mathcal{E}'(G_2(\mu)))$.

We show how to transform the given extendable SEFE $(\mathcal{E}'(G_1(\mu)), \mathcal{E}'(G_2(\mu)))$ into an extendable SEFE $(\mathcal{E}(G_1(\mu)), \mathcal{E}(G_2(\mu)))$ of $(G_1(\mu), G_2(\mu))$ such that the embedding of $\text{skel}(\mu)$ in $(\mathcal{E}(G_1(\mu)), \mathcal{E}(G_2(\mu)))$ is $\mathcal{E}(\text{skel}(\mu))$ and the SEFE of $(G_1(\mu_i), G_2(\mu_i))$ is either $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ or its flip, for each $i = 1, \ldots, k$.

If μ is an R-node, then the embedding of $\text{skel}(\mu)$ is unique up to a flip, hence $\mathcal{E}'(\text{skel}(\mu))$ and $\mathcal{E}(\text{skel}(\mu))$ coincide up to a flip of $(\mathcal{E}'(G_1(\mu)), \mathcal{E}'(G_2(\mu)))$. Moreover, by Lemma 8.5, the SEFE of $(G_1(\mu_i), G_2(\mu_i))$ can be set to be either $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ or its flip, without changing the rest of the graph. Thus, after the SEFE of $(G_1(\mu_i), G_2(\mu_i))$ is set to be either $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ or its flip, for $i = 1, \ldots, k$, the claimed SEFE is obtained.

If μ is a P-node, then an embedding of $\text{skel}(\mu)$ is a clockwise ordering of the virtual edges $e_1, \ldots, e_\ell$ of $\text{skel}(\mu)$. Consider the graph O whose vertices are $e_1, \ldots, e_\ell$ and that contains an edge (e_i, e_j) if the children of μ corresponding to e_i and e_j share an outer edge. Observe that, by Lemmas 8.2 and 8.3, e_i and e_j are adjacent in any compatible embedding of $\text{skel}(\mu)$. Since a SEFE of $(G_1(\mu), G_2(\mu))$ exists, the graph O is either a cycle or a disjoint union of paths and isolated vertices. If O is a cycle, then the clockwise ordering of $e_1, \ldots, e_\ell$ in $\mathcal{E}'(\text{skel}(\mu))$ and in $\mathcal{E}(\text{skel}(\mu))$ is the same up to a flip of $(\mathcal{E}'(G_1(\mu)), \mathcal{E}'(G_2(\mu)))$. Otherwise, denote by $O_1, \ldots, O_r$ the connected components of O and, for $i = 1, \ldots, r$, let $G_1(O_i)$, $G_2(O_i)$, and $G_{1\cap 2}(O_i)$ be the corresponding subgraphs of G_1, G_2, and $G_{1\cap 2}$, respectively. The virtual edges of $\text{skel}(\mu)$ belonging to the same connected component O_i of O form an interval both in the clockwise ordering of $e_1, \ldots, e_\ell$ defining $\mathcal{E}(\text{skel}(\mu))$ and in the clockwise ordering of $e_1, \ldots, e_\ell$ defining $\mathcal{E}'(\text{skel}(\mu))$. Hence, $\mathcal{E}'(\text{skel}(\mu))$ and $\mathcal{E}(\text{skel}(\mu))$ may differ only for the clockwise order in which the different components of O occur and for the flip of the SEFE of $(G_1(O_i), G_2(O_i))$, for each connected component O_i of O. However, for $j \neq i$, $G_1(O_i)$ and $G_2(O_i)$ share with $G_1(O_j)$ and $G_2(O_j)$ only vertices $u(\mu)$ and $v(\mu)$. Therefore, the SEFEs of $(G_1(O_1), G_2(O_1)), \ldots, (G_1(O_r), G_2(O_r))$ in $(\mathcal{E}'(G_1(\mu)), \mathcal{E}'(G_2(\mu)))$ can be ordered and independently flipped as in $\mathcal{E}(\text{skel}(\mu))$, therefore obtaining a SEFE of $(G_1(\mu), G_2(\mu))$ in which the embedding of $\text{skel}(\mu)$ is $\mathcal{E}(\text{skel}(\mu))$. Finally, by Lemma 8.5, the SEFE of $(G_1(\mu_i), G_2(\mu_i))$ can be set to be either $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ or its flip, without changing the rest of the graph. Thus, after the SEFE of $(G_1(\mu_i), G_2(\mu_i))$ is set to be either $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ or its flip, for $i = 1, \ldots, k$, the claimed SEFE is obtained. □

Theorem 8.1 suggests a very simple polynomial-time algorithm to test the existence of a SEFE of two planar graphs G_1 and G_2 whose intersection graph is biconnected.

Namely, bottom-up traverse the SPQR-tree $\mathcal{T}$ of $G_{1\cap 2}$ and compute an extendable Sefe $(\mathcal{E}(G_1(\mu)), \mathcal{E}(G_2(\mu)))$ of $(G_1(\mu), G_2(\mu))$ for each node $\mu \in \mathcal{T}$ that is not an S-node. When processing a node $\mu \in \mathcal{T}$ that is not an S-node, an extendable Sefe $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ of $G_1(\mu_i)$ and $G_2(\mu_i)$ is already fixed up to a flip for each visible node μ_i of μ.

A compatible embedding $\mathcal{E}(\text{skel}(\mu))$ of $\text{skel}(\mu)$ is then found as follows: If μ is an R-node, then $\mathcal{E}(\text{skel}(\mu))$ is (up to a flip) the only planar embedding of $\text{skel}(\mu)$; if μ is a P-node, then $\mathcal{E}(\text{skel}(\mu))$ is defined by a circular ordering $\mathcal{O}$ of the virtual edges of $\text{skel}(\mu)$ such that two virtual edges whose corresponding children of μ share an outer edge are consecutive in $\mathcal{O}$; observe that if such an ordering $\mathcal{O}$ does not exist, then G_1 and G_2 have no Sefe.

Next, we determine flips for the Sefes $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$, for each visible node μ_i of μ, and we determine for each outer edge of μ, for each internal edge of μ, for the (possible) intra-pole edge of μ, and for each internal edge of any S-node child of μ a face of $\mathcal{E}(\text{skel}(\mu))$ in which is embedded. Observe that such choices completely specify a Sefe $(\mathcal{E}(G_1(\mu)), \mathcal{E}(G_2(\mu)))$ of $(G_1(\mu), G_2(\mu))$. It is not hard to see that any internal edge of μ and of its children S-nodes, and any external edge of μ can be embedded in at most two different faces. On the other hand, this does not hold for intra-pole edges. In particular, an intra-pole edge of a P-node could possibly be embedded in a linear number of faces. The following lemma shows that intra-pole edges do not impose any additional embedding constraints and can thus be handled separately.

Lemma 8.6. *Let (G_1, G_2) be an instance of* Sefe *with $G_{1\cap 2}$ biconnected and let (G_1', G_2') be the instance obtained from (G_1, G_2) by removing all the exclusive edges that are intra-pole edges. Let $\Gamma' = (\mathcal{E}_1', \mathcal{E}_2')$ be a* Sefe *of (G_1', G_2'). Then, there exists a* Sefe *of (G_1, G_2) if and only if the intra-pole edges can be reinserted into $(\mathcal{E}_1', \mathcal{E}_2')$ without creating crossings. Moreover, the reinsertion can be done in linear time.*

Proof. Let $e^\star$ be an intra-pole edge belonging to G_1. If the endpoints of $e^\star$ share a face in $\mathcal{E}_1'$ we simply embed $e^\star$ into this face. Note that this procedure never causes a crossing between two intra-pole edges of G_1. We proceed analogously for $\mathcal{E}_2'$ and the intra-pole edges belonging to G_2. This either results in the claimed Sefe of G_1 and G_2 or we find an intra-pole edge $e^\star$ belonging to G_1 (to G_2), whose endponts do not share a common face in $\mathcal{E}_1'$ (in $\mathcal{E}_2'$).

We prove that in the latter case G_1 and G_2 do not admit a Sefe. Assume that $e^\star$ is such an intra-pole edge, belonging without loss of generality to G_1. Let μ the top-most node of the SPQR-tree $\mathcal{T}$ of $G_{1\cap 2}$ for which $e^\star$ is an intra-pole edge.

If μ is a Q-node then $e^\star$ is parallel to the edge e represented by μ and $e^\star$ can be embedded parallel to e in $\mathcal{E}_1'$.

If μ is an S-node, its parent μ' must be an R-node; otherwise we would have two adjacent S-nodes in $\mathcal{T}$ or $e^\star$ would also be an intra-pole edge of the parent. Therefore the edge $e^\star$ can be embedded in at most two possible faces f_1 and f_2 of $\text{skel}(\mu')$ with the embedding induced by $\mathcal{E}_1'$. It follows that there exist outer edges e_1 and e_2 of μ belonging to G_1 that are embedded in f_1 and f_2, respectively. The edges e_1 and e_2 are therefore either internal edges or outer edges of μ' and hence the embeddings of e_1 into f_1 and of e_2 into f_2 is forced and the endpoints of $e^\star$ do not share a face in any Sefe of G_1' and G_2', contradicting the assumption that G_1 and G_2 admit a Sefe.

If μ is an R-node, consider the embedding of $\text{skel}(\mu)$ induced by $\mathcal{E}_1'$. Again, $e^\star$ must be embedded into one of the two faces f_1 and f_2 of $\text{skel}(\mu)$ that are incident to the edge that μ shares with its parent as these are the only two faces its endpoints share in $\mathcal{E}_1'|_{G_1}$. Since neither of these choices is possible there exist outer edges e_1 and e_2 of μ, both belonging

to G_1, that are embedded in f_1 and f_2, respectively. Again there is no choice for these edges and therefore a SEFE does not exist.

Finally, if μ is a P-node, then $e^\star$ can potentially be embedded into any face of skel(μ) with the embedding induced by $\mathcal{E}'_1$. Let $e_1, \ldots, e_k$ be the virtual edges of skel(μ) in the order around $u(\mu)$ and such that e_1 is the virtual edge that μ shares with its parent. Assume that there is a pair e_i, e_{i+1} with $1 < i < k$ that are not connected by an internal edge of μ. Then there is no edge of G_1 that connects e_i to e_{i+1} and therefore the endpoints of $e^\star$ share a face in $\mathcal{E}'_1$. Analogously, the face between e_k and e_1 and the face between e_1 and e_2 must contain an outer edge of μ belonging to G_1. Again, all embedding choices for these edges are forced and therefore a SEFE of G_1 and G_2 does not exist.

Since the insertion process only requires identifying a common face of two vertices for each intra-pole edge, it can be implemented to run in linear total time. □

In the following, we therefore assume that (G_1, G_2) has no intra-pole edges. Once a SEFE for such an instance has been found possible intra-pole edges can easily be reinserted in total linear time.

We now show how to find flips of the visible nodes of μ and embeddings of the internal and outer edges of μ and of its visible nodes into faces of skel(μ) that result in an extendable SEFE of $G_1(\mu)$ and $G_2(\mu)$. Let skel$'(\mu)$ be the graph obtained from skel(μ) by replacing each virtual edge corresponding to an S-node ν with a path whose edges correspond to the children of ν. Let $\mathcal{E}$(skel$'(\mu)$) be the embedding of skel$'(\mu)$ obtained from $\mathcal{E}$(skel(μ)) by replacing each virtual edge corresponding to an S-node with its associated path.

Note that some of the exclusive edges must be embedded in a unique face, in particular all internal and outer edges of μ. Doing so possibly restricts the flips of the visible nodes containing their attachment points. Observe that constraints stemming from different edges may enforce different flips on the same visible node; in this case we conclude that a SEFE does not exist. Fixing the flip of a node may in turn restrict the possible faces for other edges. If an edge has no candidates left, we conclude that a SEFE does not exist. If it has only one face left, we embed it there, again possibly fixing the flips of the visible nodes containing its endpoints. This process stops when either all exclusive edges are embedded or each remaining visible node has two possible flips and each remaining edge has two possible faces into which it can be embedded. In the former case we can arbitrarily choose the flips of the visible nodes that have not yet been fixed and obtain an extendable SEFE of $G_1(\mu)$ and $G_2(\mu)$. For the latter case, we give a 2SAT formula whose satisfying assignments are in one-to-one correspondence with the flips and embeddings of the not-yet-fixed visible nodes and edges that yield an extendable SEFE of $G_1(\mu)$ and $G_2(\mu)$. Note that each of these nodes has two possible flips and each edge can possibly be embedded into two faces.

We introduce one variable x_ν for each visible node of μ whose flip is not yet fixed. For each exclusive edge that is not yet embedded and may be embedded in two faces f_1 and f_2 of skel$'(\mu)$ we introduce the variables $x_e^{f_1}$ and $x_e^{f_2}$. The meaning of the variables is that $x_\nu = \texttt{true}$ iff the SEFE of $(G_1(\nu), G_2(\nu))$ is $(\mathcal{E}(G_1(\nu)), \mathcal{E}(G_2(\nu)))$ and its flip otherwise. The variable x_e^f is true iff e is embedded in the face f. The formula consists of two parts. First, for each edge e that can be embedded in faces f_1 and f_2 we introduce the constraints $x_{e^*}^{f_1} \vee x_{e^*}^{f_1}$ and $\overline{x_{e^*}^{f_1}} \vee \overline{x_{e^*}^{f_1}}$ to ensure that e is embedded into exactly one of these faces. Moreover, if embedding an edge e into a face f implies a certain flip of a visible node of μ that contains an endpoint of e we express this as an implication, which is a single 2SAT clause. Analogously, we express implications that certain flips of visible nodes may have on the embedding of an edge. This part is called *consistency part* and it expresses the

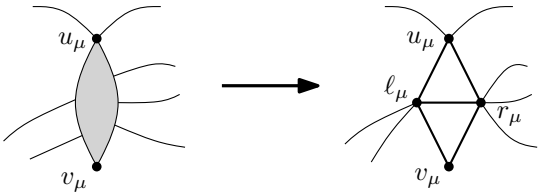


Figure 8.3.: The gadget of a node of μ that is not an S-node, represented as a thick gray edge. The thin edges attaching to μ are the exclusive edges of μ. After the transformation they are attached to either ℓ_μ or r_μ.

constraints arising from the fact that embedding an exclusive edge into a face may require a certain flip of the visible nodes containing its endpoints and vice versa. It is not hard to see that the consistency part of the formula has size linear in the size of $\text{skel}'(\mu)$ and the number of exclusive edges that need to be embedded.

The second part expresses that the resulting embedding should be planar, and is therefore called the *planarity part*. For each pair of non-embedded edges e_1 and e_2 of G_1 (of G_2) that would cross if they were embedded into the same face f we add the constraint $\overline{x_{e_1}^f} \vee \overline{x_{e_2}^f}$ to express that at least one of them must not be embedded in f. Clearly, the planarity part has polynomial (in fact at most quadratic) size.

By construction the formula is satisfiable if and only if $G_1(\mu)$ and $G_2(\mu)$ admit an extendable SEFE and such a SEFE can be constructed from a satisfying truth assignment. Since 2SAT can be solved efficiently [APT79], in fact in linear time, this yields a polynomial-time algorithm. The main bottleneck concerning the running time is that the constructed formula may be quadratic in the size of $\text{skel}'(\mu)$. We will improve on this in the next section.

8.3.2. A Linear-Time Algorithm

We now show how to improve the running time of the algorithm described in Section 8.3.1 to linear.

We start with some further definitions. First, we remark that, when a node μ is processed during the bottom-up traversal of the SPQR-tree $\mathcal{T}$ of $G_{1\cap 2}$, an embedding of the pertinent graph $G_{1\cap 2}$ of μ is fixed, up to a flip of the whole graph. This embedding determines a partition of the outer edges of μ into *left* and *right* edges, according to the position of their endpoint on the outer face, which can either lie on the counterclockwise path from $u(\mu)$ to $v(\mu)$ or on the counterclockwise path from $v(\mu)$ to $u(\mu)$. Lemma 8.4 shows that the partition into left and right edges of any node that is not an S-node is unique, although flipping the embedding swaps left and right edges.

Let μ be a node that is not an S-node with an embedding $\mathcal{E}(G_{1\cap 2}(\mu))$ that allows for an extendable SEFE of $(G_1(\mu), G_2(\mu))$. Let the *gadget* of μ be a graph with four vertices, namely its poles u_μ, v_μ and two vertices ℓ_μ and r_μ, called *attachment vertices*, and with five edges, namely $u_\mu\ell_\mu$, $u_\mu r_\mu$, $\ell_\mu v_\mu$, $r_\mu v_\mu$, and $\ell_\mu r_\mu$; see Figure 8.3. Note that the gadget of μ describes the behavior of the pertinent graph of μ when its embedding has been fixed up to a flip, in the sense that the only embedding choice for the gadget concerns its flip and, regardless of this choice, the two attachment vertices ℓ_μ and r_μ lie on opposite sides of the outer face (when considering the outer face as the union of two paths connecting u_μ and v_μ), representing the fact that the outer face of $\mathcal{E}(G_{1\cap 2}(\mu))$ has two sides (that are the paths connecting u_μ and v_μ) where the left and the right outer edges of μ can be attached.

In order to handle the process of finding an embedding of the skeleton of each non-S-node μ and of finding flips of the embeddings $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ for each visible node μ_i of μ, we introduce, for each node μ of $\mathcal{T}$, the *model* of μ, denoted by $M(\mu)$, which contains all the information that is necessary to choose an embedding of skel(μ) and the flip of $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ for each visible node μ_i of μ. The model $M(\mu)$ consists of (1) a *frame* that is composed of the gadgets of the visible nodes of μ arranged as in the skeleton of μ, and of (2) the exclusive edges that *occur in* μ, that is, the internal edges, the intra-pole edge, the outer edges of μ, and the internal edges of the S-node children of μ.

We construct the frame starting from skel(μ) and replacing each virtual edge of skel(μ) that corresponds to an S-node child ν of μ with a path of length equal to the number of children of ν. Moreover, we subdivide the virtual edge of skel(μ) representing the rest of the graph with a vertex z representing the outer face. Finally, we replace each edge corresponding to a visible node ν of μ with its gadget. Note that the model $M(\mu)$ is defined also when μ is an S-node and that the gadgets of the nodes that are children of μ appear both in $M(\mu)$ and in the model of the parent of μ. However, we will use such two models in different steps of the algorithm, in such a way that the embedding choices are coherent.

In order to explain how to handle the exclusive edges that occur in μ we need some more definitions. Let a be a vertex of G that is incident to an exclusive edge e occurring in μ. We define the *representative* of a in $M(\mu)$ as follows. If a does not belong to μ, then its representative is z. If a is a vertex of skel(μ) or a cutvertex of an S-node child of μ, then its representative is a itself. If none of the previous cases applies, then a belongs to the pertinent graph of a unique visible node ν of μ. In this case e is an outer edge of ν and therefore a lies on the outer face of any embedding of $G_{1\cap 2}(\nu)$ that allows for an extendable SEFE of $G_1(\nu)$ and $G_2(\nu)$. If a lies on the clockwise path from $u(\nu)$ to $v(\nu)$ along the outer face of an embedding of $G_{1\cap 2}(\mu)$, then its representative is ℓ_ν, otherwise its representative is r_ν. Note that the partition of the outer edges of ν into those having ℓ_ν as representative and those having r_ν as representative is unique by Lemma 8.4, and it does not depend on the actually chosen embedding of $G_{1\cap 2}(\nu)$. Flipping the chosen embedding of $G_{1\cap 2}(\nu)$ maintains the same partition but swaps ℓ_ν with r_ν.

We now add the exclusive edges occurring in μ to the model $M(\mu)$. For any exclusive edge (u, v) of G_1 or of G_2 occurring in μ we add to $M(\mu)$ the edge between the representatives of its endvertices. Figure 8.4 shows the model of the node μ depicted in Figure 8.2.

A SEFE of a model $M(\mu)$ is an embedding of the model such that crossings only occur between pairs of exclusive edges where one edge stems from G_1 and the other one from G_2. Notice that there is a one-to-one correspondence between the SEFEs of $M(\mu)$ and the extendable SEFEs of $G_1(\mu)$ and $G_2(\mu)$. Namely, an embedding of the frame of μ corresponds to an embedding of skel(μ) plus a possible flip of the SEFE $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$, for each visible node μ_i for which a SEFE $(\mathcal{E}(G_1(\mu_i)), \mathcal{E}(G_2(\mu_i)))$ has already been decided; an embedding in $M(\mu)$ of the edges that occur in μ corresponds to an embedding in an extendable SEFE of $(G_1(\mu), G_2(\mu))$ of the edges that occur in μ. Moreover, if $(G_1(\mu), G_2(\mu))$ has an extendable SEFE $(\mathcal{E}(G_1(\mu)), \mathcal{E}(G_2(\mu)))$, then the same embedding and flipping choices lead to a SEFE of $M(\mu)$. The converse is, in general, not true. In fact, $M(\mu)$ may allow for a SEFE, while the same embedding and flip choices do not lead to an extendable SEFE of $(G_1(\mu), G_2(\mu))$. However, the next lemma shows that in this case $(G_1(\mu), G_2(\mu))$ do not allow for an extendable SEFE at all. Hence, once a SEFE of $M(\mu)$ has been determined for each μ, the algorithm has to check whether the resulting embedding for $G_{1\cap 2}$ allows for an extendable SEFE.

Lemma 8.7. *Let G_1 and G_2 be two planar graphs whose intersection graph $G_{1\cap 2}$ is*

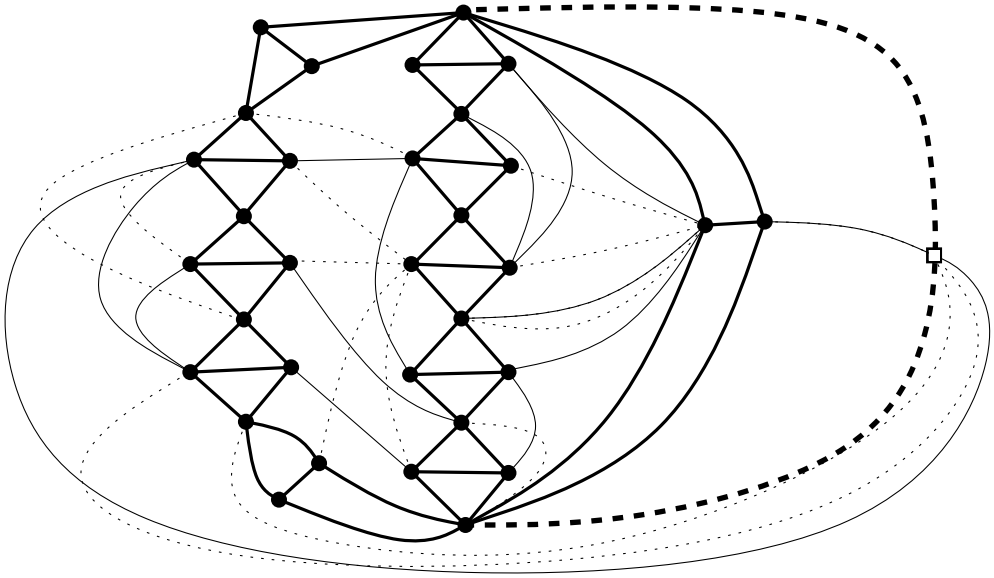


Figure 8.4.: The model of the node μ shown in Figure 8.2.

biconnected. Let μ be a non-S-node of the SPQR-tree $\mathcal{T}$ of $G_{1\cap 2}$, let $M(\mu)$ be the model of μ, and let $\mathcal{E}_{M(\mu)}$ be a Sefe *of $M(\mu)$. Suppose that (G_1, G_2) has a* Sefe*. Then the embedding and the flipping choices induced by $\mathcal{E}_{M(\mu)}$ lead to an extendable* Sefe *of $(G_1(\mu), G_2(\mu))$.*

Proof. Assume, for a contradiction, that the embedding of $(G_1(\mu), G_2(\mu))$ (together with the edges towards the outer face) induced by $\mathcal{E}_{M(\mu)}$ has a crossing. Observe that such a crossing can only involve two edges e_1 and e_2 that share an attachment vertex in $M(\mu)$, but do not share any vertex in $(G_1(\mu), G_2(\mu))$, as otherwise such a crossing would appear in $M(\mu)$ as well. Then, such edges are both incident to the same path that connects $u(\mu_i)$ and $v(\mu_i)$ along the outer face of the embedding of $(G_1(\mu_i), G_2(\mu_i))$, for some visible node μ_i of μ. First, this rules out the possibility that one of e_1 and e_2 is an intra-pole edge of μ. Second, if one of the crossing edges is an internal or an outer edge of μ, then the face of the embedding of skel(μ) into which it has to be embedded is fixed. By Lemma 8.4, the order of the attachment vertices of all these edges around this face is fixed (up to a flip) and therefore the crossing occurs in every embedding of $(G_1(\mu), G_2(\mu))$, thus contradicting the assumption that (G_1, G_2) admits a Sefe. Third, if both the endpoints of e_1 and both the endpoints of e_2 belong to the same S-node child ν of μ, then there are two faces into which e_1 and e_2 can be embedded. Clearly, since both have endpoints on the same path from $u(\mu_i)$ to $v(\mu_i)$, the two edges must be embedded in the same face. Again, by Lemma 8.4, for each child μ_i of ν the clockwise order of attachment vertices of e_1 and e_2 is the same up to reversal in each of the nodes μ_i, but also along the outer face of ν, as the order of the children of ν is fixed. Hence, e_1 and e_2 cross in every embedding of $(G_1(\mu), G_2(\mu))$, thus contradicting the assumption that (G_1, G_2) admits a Sefe. □

Lemma 8.7 shows that, in order to find an extendable Sefe of $G_1(\mu)$ and $G_2(\mu)$, if one exists, it is enough to deal with the models of the non-S-nodes of the SPQR-tree of $G_{1\cap 2}$. Moreover, note that it is not necessary to keep track of multiple edges of $M(\mu)$ stemming from the same graph out of G_1 and G_2. Hence, even if a linear number of nodes of $\mathcal{T}$ might exist, each with a linear number of outer edges, $M(\mu)$ only contains $O(|M(\mu)|)$ such edges

for each μ, and hence the total size of all the models is linear.

The linear-time implementation requires two ingredients. First, we need to compute in linear total time the model of each node of $\mathcal{T}$. Second, we have to show that, given the model $M(\mu)$ of a node μ that is not an S-node, it is possible to compute a SEFE of $M(\mu)$ in time linear in the size of $M(\mu)$.

For the computation of the models we require a data structure that, given a node μ and a vertex v, allows us to find the vertex or the virtual edge representing v in the skeleton of $G_{1\cap2}(\mu)$.

Lemma 8.8. *Let G be an n-vertex biconnected planar graph and let $\mathcal{T}$ be its SPQR-tree. When traversing $\mathcal{T}$ in bottom-up order, it is possible to maintain in total $O(n)$ time a data structure that, for every node μ of $\mathcal{T}$, allows to query for a given vertex v of G the representative of v in μ in constant time.*

Proof. This can be done with a simple application of a union-find data structure. The main observation that is required to achieve total $O(n)$ time is that the sequence of union operations only depends on $\mathcal{T}$ and is therefore known in advance. Hence, the $O(n)$-time version of union-find by Gabow and Tarjan [GT85] applies. □

We now show how to compute the models of all the non-S-nodes of $\mathcal{T}$ in total $O(n)$ time.

Lemma 8.9. *Let G_1 and G_2 be two n-vertex planar graphs whose intersection graph $G_{1\cap2}$ is biconnected and let $\mathcal{T}$ be the SPQR-tree of $G_{1\cap2}$. It is possible to compute in total $O(n)$ time the models of all the non-S-nodes of $\mathcal{T}$ or to conclude that (G_1, G_2) does not admit any* SEFE.

Proof. Let $\mathcal{T}$ be rooted at an arbitrary Q-node. First, observe that the frame of each non-S-node $\mu \in \mathcal{T}$ can be easily computed by replacing each virtual edge representing a visible node ν of μ in skel(μ) with the gadget of ν and by replacing the virtual edge representing the rest of the graph with a path composed of two edges. Since the total number of virtual edges in $\mathcal{T}$ is $O(n)$ and since each gadget has constant size, the frames of all the non-S-nodes of $\mathcal{T}$ can be computed in total $O(n)$ time.

Let uv be an exclusive edge of G_1 or of G_2. Suppose that uv is an outer edge of some node μ of $\mathcal{T}$. Then, one of its endvertices, say u, belongs to $G_{1\cap2}(\mu)$, while v does not; moreover, u is not a pole of μ.

We define the *first node* of u, denoted by $\mathcal{F}(u)$, as the lowest node of $\mathcal{T}$ such that $G_{1\cap2}$ contains u and u is not a pole of $\mathcal{F}(u)$. Observe that $\mathcal{F}(u)$ is the lowest common ancestor of all the Q-nodes of $\mathcal{T}$ that represent edges incident to u. Further, any node μ of $\mathcal{T}$ that has uv as an outer edge, with u in $G_{1\cap2}(\mu)$, lies on the path between $\mathcal{F}(u)$ and the root of $\mathcal{T}$. Similarly, any node μ that contains uv as an outer edge, with v in $G_{1\cap2}(\mu)$, lies on the path between $\mathcal{F}(v)$ and the root of $\mathcal{T}$. Finally, edge uv is an internal edge only in the lowest node of $\mathcal{T}$ whose pertinent graph contains both u and v. Assuming that neither u nor v is an endvertex of the edge at which $\mathcal{T}$ is rooted, such a node coincides with the lowest common ancestor of $\mathcal{F}(u)$ and $\mathcal{F}(v)$, denoted by $\mathcal{I}(u, v)$.

Hence, edge uv appears as an outer edge at the first nodes $\mathcal{F}(u)$ and $\mathcal{F}(v)$ of u and v, respectively, and it continues to appear as an outer edge when processing their parents until the lowest common ancestor $\mathcal{I}(u, v)$ of $\mathcal{F}(u)$ and $\mathcal{F}(v)$ is considered, where uv occurs as an internal edge. Note that it may happen that two or three of the nodes $\mathcal{F}(u)$, $\mathcal{F}(v)$, and $\mathcal{I}(u, v)$ coincide. If $\mathcal{F}(u)$ coincides with $\mathcal{I}(u, v)$ and $\mathcal{F}(v)$ does not, edge uv does not

appear as an outer edge of $\mathcal{F}(u)$, while it appears as an outer edge of each node on the path between $\mathcal{F}(v)$ and $\mathcal{I}(u,v)$, excluding the last node of this path. If $\mathcal{F}(u)$, $\mathcal{F}(v)$, and $\mathcal{I}(u,v)$ coincide, then uv does not occur as an outer edge of any node of $\mathcal{T}$. To handle the case in which u is an endvertex of the edge e at which $\mathcal{T}$ is rooted, we simply exclude the Q-node representing e when computing the first node of u. Note that an edge uv does not necessarily occur as an internal edge in $\mathcal{I}(u,v)$. Namely, if the endpoints of uv are both vertices of $\text{skel}(\mathcal{I}(u,v))$ and $\text{skel}(\mathcal{I}(u,v))$ contains the virtual edge uv, then e is an intra-pole edge of the child of $\text{skel}(\mathcal{I}(u,v))$ corresponding to the virtual edge uv. Notice that, in such a case, uv is not an internal edge and is not an outer edge for any node of $\mathcal{T}$.

We now describe how we can exploit the previous observations to quickly compute the models of all the non-S-nodes of $\mathcal{T}$ in a bottom-up traversal of $\mathcal{T}$. Note that, using Harel and Tarjan's lowest common ancestor data structure [HT84], we can compute the first vertex $\mathcal{F}(u)$ of each vertex u in total $O(n)$ time. Analogously, $\mathcal{I}(u,v)$ can be computed for all the exclusive edges in total $O(n)$ time. Each processed node of $\mathcal{T}$ whose parent has not yet been processed stores a partition of its outer edges into two linked lists L_μ and R_μ containing its left and right exclusive edges, respectively. These lists are implemented intrusively, in such a way that each edge appears in two lists, one for each endpoint, and any given edge can be removed in $O(1)$ time from a list, without the need of finding its location inside the list. We initialize the lists of all the Q-nodes to an empty list. In the following we show how to construct L_μ and R_μ in total $O(n)$ time for all the nodes μ of $\mathcal{T}$.

To distribute the exclusive edges to the lists L_μ and R_μ, we maintain several lists. For each node μ we keep a list F_μ of all the edges that have an endpoint whose first node is μ and a list I_μ of all the edges uv for which $\mathcal{I}(u,v)$ is μ, that is, uv possibly occurs as an internal edge of μ. Observe that, given that the first vertex $\mathcal{F}(u)$ of each vertex u can be computed in total $O(n)$ time and given that $\mathcal{I}(u,v)$ can be computed for all the exclusive edges in total $O(n)$ time, lists F_μ and I_μ can be constructed in total $O(n)$ time for all the nodes μ of $\mathcal{T}$.

To process a node μ that is not an S-node, we first show how to construct the model $M(\mu)$. In a second step we compute its lists L_μ and R_μ. Recall that the frame of μ has already been computed; then initialize the model $M(\mu)$ to be the frame of μ. Next, we identify the exclusive edges occurring in μ, except for the intra-pole edges of μ, which will be identified in a subsequent step.

Essentially, for each edge $e = uv$ that occurs in μ, we need to find the representatives of its endpoints in the model. We could identify all internal edges by traversing the lists $\mathcal{I}_\mu$ and $\mathcal{I}_{\mu'}$ for each S-node child μ' of μ. While the data structure from Lemma 8.8 allows us to quickly find the visible child ν to which an endpoint u of an edge e belongs, it does not help us in determining whether its representative is ℓ_ν or to r_ν. Basically, to find out this information, we would like to traverse each of the lists L_ν and R_ν for all visible nodes ν of μ. Edges $e = uv$ in the list L_ν have ℓ_ν as the representative of their attachment in ν, and analogously edges in R_ν have r_ν as representatives of their attachment in ν. However, it is also not possible to simply traverse all these lists, as they may contain, in addition to the internal edges, a linear number of external edges. And since a linear number of skeletons may have a linear number of external edges, this would result in quadratic total time.

Instead, we use the following observations in order to quickly identify the representatives of all exclusive edges. First, for each visible node ν of μ only one of the vertices ℓ_ν and r_ν can be incident to the outer face in any embedding of $M(\mu)$. Therefore, if $M(\mu)$ admits a SEFE, at most one of the lists L_ν and R_ν may contain external edges. We can therefore

afford to traverse the list R_ν, and thus identify the representative vertices of endpoints of internal edges, until we meet the first external edge in R_ν. We then stop the traversal of R_ν and do the same with L_ν. During the traversal we remove from the lists all edges that we process. If both traversals are aborted due to an occurrence of an external edge, then $M(\mu)$ does not admit a SEFE, and we reject the instance. Therefore, after performing this step for every visible node ν of μ, we have either rejected the instance, or, for each visible node ν of μ one of the lists L_ν and R_μ has become empty. Second, in this case, we know that all further edges that may attach to ν must attach to the side whose list is not yet empty and has possibly not been traversed completely. Thus, now we can use the lists $\mathcal{I}_\mu$ and $\mathcal{I}_{\mu'}$ for S-node children μ' of μ and for all S-node children of ν, to find all the exclusive edges that occur in $M(\mu)$, except for the external edges. Namely, when traversing these lists, the representative of each endpoint of an edge $e = uv$ has either been previously determined, or using Lemma 8.8, the corresponding visible child ν of μ can be found quickly, and by determining the non-empty list of this node, the representative can be identified in constant time. Note that each edge belonging to one of such lists is an internal edge of a visible node or an S-node child μ' of μ, an internal edge of a visible node μ, or an intra-pole edge of one of the visible nodes of μ. When we find an intra-pole edge of a visible node ν of μ that is not a an S-node, we pass it down to the model $M(\nu)$.

We remove each edge that we process in such a way from all the remaining lists it is contained in, in particular from the lists L_ν and R_ν of any child ν (recall that each edge can be contained in only one of such lists for each endpoint, thus the removal can be done in constant time). Hence, after this step, the only edges remaining in any of the lists L_ν and R_ν are outer edges. Note that it is enough to know for each visible node whether it has any outer edges and on which side they are, rather than to know their exact number. Therefore, we insert only one of the remaining edges from ν to z in $M(\mu)$ if the lists L_ν and R_ν are not both empty.

Finally, we traverse the lists F_ν and F_μ to find the remaining edges starting at vertices of the model. Again, the edges identified in this way may be intra-pole edges of a visible node, in which case they are passed down to the corresponding model. Otherwise, we add them either as an internal or as an outer edge to our model. If an edge is internal, we remove it from F_ν or F_μ, so that after this step only outer edges remain in these lists. Observe that, after this step, we have identified all the edges of the model, except for possibly an intra-pole edge, which might be added at a later step.

It remains to show how to construct L_μ and R_μ. First, we find a compatible embedding of skel(μ). For an R-node this amounts to checking whether all the visible nodes that have a non-empty list L_μ or R_μ and all the vertices v incident to edges in $\mathcal{F}_\mu$ are incident to the outer face. If this is not possible, then the instance does not admit a SEFE by Lemma 8.3. For a P-node we reorder the virtual edges such that all the visible nodes with non-empty lists L_μ or R_μ are adjacent to the outer face. If this is not possible, then the instance does not admit a SEFE, by Lemma 8.3. To compute L_μ we traverse the boundary of the outer face from u_μ to v_μ in counterclockwise order and concatenate all the non-empty lists of visible nodes and vertices encountered on the way. We compute R_μ analogously by a traversal along the counterclockwise boundary of the outer face from v_μ to u_μ.

It is not hard to see that the running time is linear in the size of the model and in the sizes of I_μ, F_μ, and of F_ν, for each visible node ν of μ. The total linear time follows from the fact that each exclusive edge e occurs only in one of the lists I_μ and in two of the lists F_μ of the whole SPQR-tree. □

Now that we have computed the models of all non-S-nodes, it remains to find a SEFE

for each of them. We show that for each model this can be done in time proportional to its size.

Lemma 8.10. *Given a model $M(\mu)$ of a non-S-node μ of the SPQR tree $\mathcal{T}$ of $G_{1\cap 2}$, it is possible to find a* SEFE *of $M(\mu)$ or to decide that no* SEFE *of $M(\mu)$ exists in $O(|M(\mu)|)$ time.*

Proof. We remove all intra-pole edges from the model and reinsert them afterwards, as described in Lemma 8.6. This step ensures that each exclusive edge can be embedded into at most two different faces of the model. The first step is to find a compatible embedding of skel(μ).

If μ is an R-node, we simply check for each exclusive edge whether its attachment points share a common face.

If μ is a P-node, we first build two auxiliary graphs O_1 and O_2, both containing one vertex for each virtual edge of skel(μ). Two vertices of O_i, with $i = 1, 2$, are connected by an edge if there is an exclusive edge in G_i connecting two vertices belonging to the nodes represented by the corresponding virtual edges. Let O be the union of O_1 and O_2. A compatible embedding exists if and only if O is either a collection of disjoint paths or a single cycle. Once O has been constructed, this can easily be checked, and a corresponding embedding can be constructed, in $O(|\text{skel}(\mu)|)$ time.

In both cases the checking requires time proportional to the size of the skeleton and to the number of exclusive edges traversing it.

Next, we describe how to flip the visible nodes in order to obtain a planar embedding of G_1 and G_2. To this end, we fix a *default flip* for each visible node of μ and construct a 2SAT formula φ_μ similar to the previous section. In fact, we introduce the same variables x_ν for all visible nodes of μ and variables x_e^f with the same meaning as before, that is, $x_\nu =$ TRUE means that ν must be flipped, while $x_\nu =$ FALSE means that ν keeps its default flip. Moreover, we also add the consistency part of the previous formula to handle the implications between embeddings of edges into faces and flips of visible nodes containing their endpoints.

Again our goal is to construct φ_μ in such a way that φ_μ is satisfiable if and only if the visible nodes of μ can be flipped to planarly embed all the exclusive edges. Namely, φ_μ is satisfiable if and only if there exists a flip of all the visible nodes such that for every exclusive edge both the attachment points share a common face and no two exclusive edges of a child S-node cross. Further, given a satisfying assignment of φ_μ, a flipping of the visible nodes that allows a planar embedding of all the exclusive edge can be found by considering the default flip for the nodes whose variable has been assigned the value FALSE and the other flip for the nodes whose variable has been assigned the value TRUE. Note that it may still be possible that internal edges of μ cross. However, we will see that in this case these crossings cannot be avoided.

To construct the planarity part of formula φ_μ, we consider the exclusive edges occurring in μ. First, we consider exclusive edges connecting vertices that belong to different virtual edges. The embedding of such an edge e is restricted to a unique face f and we simply add the clause x_e^f to encode this.

Second, we consider the exclusive edges occurring in one of the S-node children of μ. For such edges, we have to ensure that there is no crossing among them. Of course we could pick all pairs that would produce a crossing if they were embedded in the same face f and explicitly specify clauses that require them to be embedded in different faces. However, this would again result in a formula of quadratic size. Therefore, in order to get a linear-time

algorithm, we express the constraints on the flips stemming from the internal and the outer edges of ν via the model $M(\nu)$, as described below. Note that the models $M(\nu)$ of all the S-node children ν of μ can be computed in $O(|M(\mu)|)$ time from $M(\mu)$.

We show how to incorporate the constraints stemming from G_1. Let ν be an S-node child of μ. We take the model $M(\nu)$ and remove from it all the exclusive edges stemming from G_2, note that this since G_1 is planar, this yields a planar graph. Now we consider the SPQR-tree of this graph. Since each gadget is triconnected, we have that in any node of this SPQR-tree either the five edges of a gadget are represented by five different edges of the skeleton, that is, the gadget is *part of the R-node*, or they are represented by one single virtual edge. Note that gadgets that are part of the same R-node need to be flipped consistently. We therefore pick in each R-node a representative gadget and express the consistency requirements for all the other gadgets that are part of the same R-node with respect to this representative. We assume that the embedding of each R-node is chosen such that the representative ν' has its default flip. Let ν'' be any visible node whose gadget occurs in the same R-node as ν'. If ν'' maintains its default flip when ν' has its default flip, then we add the two clauses $(x_{\nu'} \vee \overline{x_{\nu''}})$ and $(\overline{x_{\nu'}} \vee x_{\nu''})$, expressing that these variables must be assigned either both TRUE or both FALSE. Analogously, if ν'' does not maintain its default flip when ν' has its default flip, we add the clauses $(x_{\nu'} \vee x_{\nu''})$ and $(\overline{x_{\nu'}} \vee \overline{x_{\nu''}})$, expressing that exactly one of them needs to be flipped. We do this for any R-node in the auxiliary graph and thus we add a number of 2SAT clauses to φ_μ that is linear in the number of children of the S-node ν, while encoding all planarity constraints stemming from ν. We incorporate the constraints stemming from G_2 analogously and do the same for all the S-node children of μ.

The resulting formula φ_μ is linear in the size of $M(\mu)$ and can be constructed in $O(|M(\mu)|)$ time. Satisfiability of φ_μ and, in the positive case, a corresponding assignment and the resulting flips can be found in $O(|\varphi_\mu|)$ time [APT79]. Clearly, if φ_μ is not satisfiable, the constraints contradict each other and a SEFE does not exist. Otherwise, since φ_μ encodes all the planarity constraints that can be met by flipping visible nodes, we can conclude that if a planar embedding of $G_1(\mu)$ and $G_2(\mu)$ exists, then there is also one with the chosen flips, that is, with the chosen embedding of $G(\mu)$ up to a flip. □

To solve the SEFE problem for graphs G_1 and G_2 with biconnected intersection graph, we now perform three steps. First, we compute the SPQR-tree $\mathcal{T}$ of $G_{1\cap 2}$ and for each non-S-node μ of $\mathcal{T}$ we compute in linear time its model $M(\mu)$, using Lemma 8.9. Next, we construct embeddings for all models, each in time proportional to the model size using Lemma 8.10. As the total size of all the models is linear, this step takes linear time, too. Finally, we construct from the simultaneous embeddings of the models an embedding of $G_{1\cap 2}$ that allows for a SEFE of G_1 and G_2, if they admit one. Given this embedding of $G_{1\cap 2}$ we can then construct the actual SEFE by using the algorithm from Chapter 6. In fact, a much simpler algorithm can be used, as the SEFE can be completely obtained from the embeddings of the models as they not only specify the embedding of the intersection graph but also for each exclusive edge the face into which it is embedded.

Theorem 8.2. *Let G_1 and G_2 be two planar graphs whose intersection graph $G_{1\cap 2}$ is biconnected. It can be checked in linear time whether G_1 and G_2 admit a* SEFE. *In this case a corresponding* SEFE *can be computed in the same time.*

8.4. The Intersection Graph is Connected

In this section we show that the Sefe problem, when the intersection graph is connected, is equivalent to a 2-page book embedding problem defined in the following.

Let G be a graph, let (E_1, E_2) be a partition of its edge set, and let T be a rooted tree whose leaves are the vertices of G. Problem Partitioned T-coherent 2-page book embedding with input (G, E_1, E_2, T) asks: Does a 2-page book embedding of G exist in which the edges of E_1 lie in one page, the edges of E_2 lie in the other page, and, for every internal vertex $t \in T$, the vertices of G in the subtree of T rooted at t appear consecutively in the vertex ordering of G defined in the book embedding?

We now show how to transform an instance $G_1 = (V, E_1), G_2 = (V, E_2)$ of Sefe in which $G_{1\cap 2}$ is connected into an instance of Partitioned T-coherent 2-page book embedding. This transformation consists of three steps.

In the first step, we transform an instance (G_1, G_2) of Sefe such that $G_{1\cap 2}$ is connected into an equivalent instance (G'_1, G'_2) of Sefe such that $G'_{1\cap 2}$ is a tree. To this end, start with graphs G'_1 and G'_2 having the same vertex set as G_1 and G_2, and having no edges. Partition the edges of $G_{1\cap 2}$ into two sets $E'_{1\cap 2}$ and $E''_{1\cap 2}$, in such a way that the edges of $E'_{1\cap 2}$ induce a spanning tree of $G_{1\cap 2}$ and the edges of $E''_{1\cap 2}$ are the edges in $G_{1\cap 2}$ not in $E'_{1\cap 2}$. The edges of $E'_{1\cap 2}$ belong to $G'_{1\cap 2}$, that is, they are added both to G'_1 and to G'_2. Also, add to G'_1 each exclusive edge of G_1 and add to G'_2 each exclusive edge of G_2. Next, for each edge uv in $E''_{1\cap 2}$, introduce new vertices u', u'_1, u'_2, v', v'_1, and v'_2 in G'_1 and in G'_2, and introduce edges uu', $u'u'_1$, $u'u'_2$, vv', $v'v'_1$, $v'v'_2$ in $G'_{1\cap 2}$, that is, add these edges both to G'_1 and to G'_2. Finally, add the exclusive edge $u'_1v'_1$ to G'_1 and the exclusive edge $u'_2v'_2$ to G'_2; see Figure 8.5. The following lemma states that these two instances are equivalent.

Lemma 8.11. *The pair (G_1, G_2) is a positive instance of Sefe if and only if (G'_1, G'_2) is a positive instance of Sefe. Moreover, (G'_1, G'_2) is such that the intersection graph $G'_{1\cap 2}$ is a tree with $O(n)$ vertices.*

Proof. We prove that $G'_{1\cap 2}$ is a tree with $O(n)$ vertices. The edges of $E'_{1\cap 2}$ induce a spanning tree of $G_{1\cap 2}$. Moreover, the edges that are introduced in $G'_{1\cap 2}$ for each edge in $E''_{1\cap 2}$ induce two trees spanning the newly introduced vertices and each one attached to exactly one vertex of $E'_{1\cap 2}$. Since the number of vertices and edges introduced for each edge in $E''_{1\cap 2}$ is constant, it follows that the intersection graph $G'_{1\cap 2}$ has $O(n)$ vertices. Next, we prove that G_1, G_2 is a positive instance of Sefe if and only if G'_1, G'_2 is a positive instance of Sefe.

First, suppose that G_1, G_2 is a positive instance of Sefe. Consider any Sefe Γ of (G_1, G_2). Construct a Sefe Γ' of G'_1, G'_2 as follows; see Figure 8.5. The edges that are common to G'_1, G'_2 and to G_1, G_2 (that is, the edges in $E'_{1\cap 2}$ plus the exclusive edges of G_1 and G_2) have the same drawing in Γ' as they have in Γ. The edges uu', $u'u'_1$, $u'u'_2$, vv', $v'v'_1$, $v'v'_2$, $u'_1v'_1$, and $u'_2v'_2$ that replace the edge uv of $E''_{1\cap 2}$ have a drawing that is arbitrarily close to the one of uv. As edge uv does not intersect any other edge in Γ, its corresponding edges in Γ' do not have any crossings, and hence Γ' is a Sefe of G'_1 and G'_2.

Second, suppose that (G'_1, G'_2) is a positive instance of Sefe. Consider any Sefe Γ' of G'_1, G'_2. For each edge uv in $E''_{1\cap 2}$ consider the drawing of the edges uu', $u'u'_1$, $u'u'_2$, vv', $v'v'_1$, $v'v'_2$, $u'_1v'_1$, and $u'_2v'_2$ in Γ'. Since edges uu', $u'u'_1$, $u'u'_2$, vv', $v'v'_1$, and $v'v'_2$ are inclusive edges of G'_1 and G'_2, they do not cross any other edge in Γ'. On the other hand, edge $u'_1v'_1$ (respectively $u'_2v'_2$) could cross exclusive edges of G'_2 (respectively of G'_1).

We show that Γ' can be modified into a Sefe of (G'_1, G'_2) such that $u'_1v'_1$ does not cross any exclusive edge of G'_2, except possibly for $u'_2v'_2$, and such that $u'_2v'_2$ does not cross any

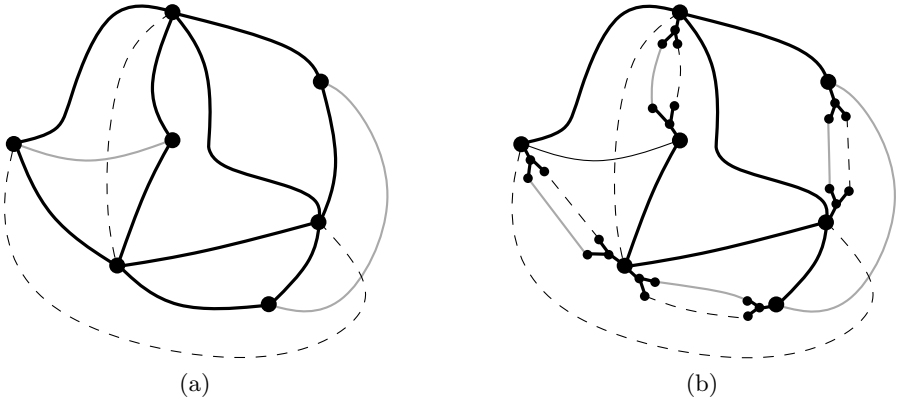


Figure 8.5.: Reduction of a SEFE instance (G_1, G_2) whose intersection graph is connected to an instance (G'_1, G'_2) whose intersection is a tree. (a) A SEFE Γ of G_1, G_2. (b) A SEFE Γ' of G'_1, G'_2 constructed from Γ.

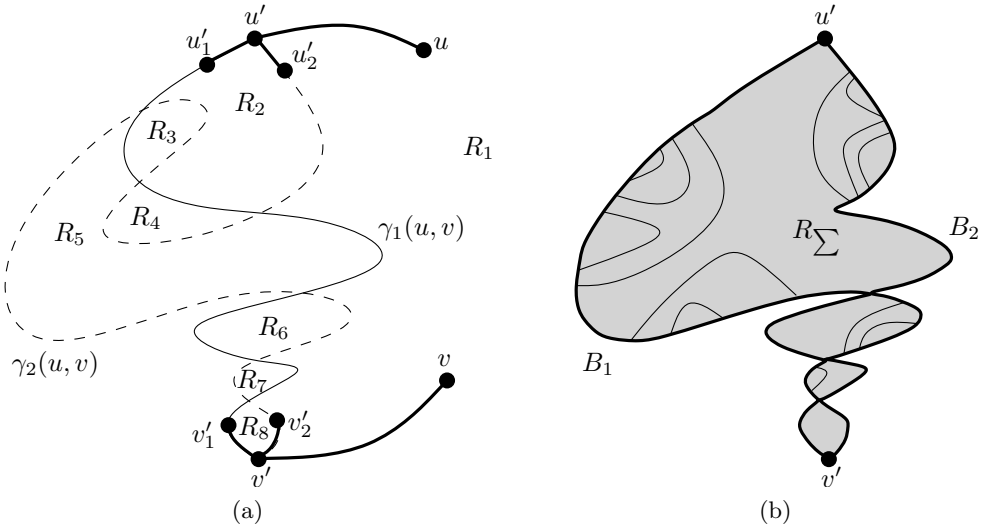


Figure 8.6.: Rerouting of the curves of a SEFE of the modified instance (G'_1, G'_2) in order to obtain a SEFE of the original instance (G_1, G_2). (a) Curves $\gamma_1(u, v)$ and $\gamma_2(u, v)$ and regions $R_1, R_2, \ldots, R_k$. (b) Region $R_\sum$, bounding curves B_1 and B_2, and the curves in S.

exclusive edge of G'_1, except possibly for $u'_1v'_1$. Consider the curve $\gamma_1(u, v)$ composed of the inclusive edges $u'u'_1$ and $v'v'_1$ and of the exclusive edge (u'_1, v'_1). Also, consider the curve $\gamma_2(u, v)$ composed of the inclusive edges $u'u'_2$ and $v'v'_2$ and of the exclusive edge $u'_2v'_2$; see Figure 8.6(a). These curves subdivide the plane into closed regions $R_1, R_2, \ldots, R_k$. Observe that both u and v belong to the same region, say R_1, as otherwise the path connecting u and v composed of edges in $E'_{1\cap2}$ would intersect $\gamma_1(u, v)$ or $\gamma_2(u, v)$, thus contradicting the assumption that Γ' is a SEFE of (G'_1, G'_2). Also, observe that both u and v are incident to R_1, as otherwise the path composed of edge uu', of edge vv', and of the path connecting u and v composed of edges in $E'_{1\cap2}$ would intersect $\gamma_1(u, v)$ or $\gamma_2(u, v)$, thus contradicting the assumption that Γ' is a SEFE of (G'_1, G'_2). Then, denote by $R_\sum$ the closed region $\bigcup_{i=2}^{k} R_i$; see Figure 8.6(b). The border of $R_\sum$ consists of a line segment B_1 clockwise connecting u' to v' and of line segment B_2 clockwise connecting v' to u'. Observe

that $R_\sum$ contains no vertex w different from u', u'_1, u'_2, v', v'_1, and v'_2, as otherwise the path from w to u that is composed of edges in $E'_{1\cap 2}$ would cross $\gamma_1(u,v)$ or $\gamma_2(u,v)$, thus contradicting the assumption that Γ' is a SEFE of (G'_1, G'_2). Since u', u'_1, u'_2, v', v'_1, and v'_2 have no incident exclusive edge other than $u'_1v'_1$ and $u'_2v'_2$, it follows that the intersection of $R_\sum$ with the exclusive edges of G'_1 different from $u'_1v'_1$ is a set S of curves whose endpoints are on the border of $R_\sum$. If a curve s in S has one endpoint in B_1 and the other endpoint in B_2 then s crosses $\gamma_1(u,v)$, thus contradicting the assumption that Γ' is a SEFE of G'_1 and G'_2. Hence, all the curves in S have both endpoints on B_1 or both endpoints on B_2. If a curve having both endpoints on B_1 exists, then there exists a curve s having both endpoints u_s and v_s on B_1, and such that the open curve that is the part $B_1(u_s, v_s)$ of B_1 between u_s and v_s contains no endpoint of a curve in S. Then, s can be replaced by a curve lying in the interior of R_1 arbitrarily close to B_1. Since no exclusive edge of G'_1 cuts $B_1(u_s, v_s)$, the resulting drawing is still a SEFE of G'_1, G'_2 with one less line segment in S. Thus, iterating such a modification we eventually get a SEFE of (G'_1, G'_2) in which no exclusive edge of G'_1 different from $u'_1v'_1$ intersects B_1. Modifying analogously the curves in S intersecting B_2 and modifying analogously the drawings of the exclusive edges of G'_2 different from $u'_2v'_2$, we eventually get a SEFE of (G'_1, G'_2) in which no exclusive edge of G'_1 and G'_2 different from $u'_1v'_1$ and from $u'_2v'_2$ crosses $R_\sum$; hence in such a SEFE no exclusive edge of G'_1 and G'_2 different from $u'_1v'_1$ and from $u'_2v'_2$ crosses $u'_1v'_1$ or $u'_2v'_2$.

After the described modification has been done for all the edges corresponding to an edge uv in $E''_{1\cap 2}$, we obtain a SEFE Γ'' of (G'_1, G'_2) and we draw each edge uv as the concatenation of the drawings of edges uu', $u'u'_1$, $u'_1v'_1$, v'_1v', $v'v$ in Γ''. Observe that no edge of G'_1 nor of G'_2 crosses uu', $u'u'_1$, v'_1v', $v'v$, since Γ'' is a SEFE. Moreover, no exclusive edge of G'_1 nor of G'_2 different from $u'_2v'_2$ crosses $u'_1v'_1$ by the construction of Γ''. Hence, by removing from Γ'' vertices u', u'_1, u'_2, v', v'_1, and v'_2 and removing edges uu', $u'u'_1$, $u'u'_2$, vv', $v'v'_1$, $v'v'_2$, $u'_1v'_1$, and $u'_2v'_2$ we get a SEFE Γ of (G_1, G_2). □

In the second step, we transform instance (G_1, G_2) of SEFE into an equivalent instance (G'_1, G'_2) of SEFE such that $G'_{1\cap 2}$ is a tree and all the exclusive edges of G'_1 and of G'_2 are incident only to leaves of $G'_{1\cap 2}$. To this end, we modify every edge $uv \in G_{1\setminus 2}$ such that u is not a leaf of $G_{1\cap 2}$ as follows. We subdivide edge uv with a new vertex u', and we add the edge uu' to E_2, so that u' is a leaf in the intersection graph of the two modified graphs. Symmetrically, we subdivide every edge $uv \in G_{2\setminus 1}$ such that u is not a leaf of $G_{1\cap 2}$ with a new vertex u', and add edge uu' to E_1, again making u' a leaf in the intersection graph of the two modified graphs. Note that the exclusive edges of G_1 and G_2 that are incident to two non-leaf vertices are subdivided twice. Denote by G'_1 and by G'_2 the resulting graphs. We have the following.

Lemma 8.12. *The pair (G_1, G_2) is a positive instance of* SEFE *if and only if (G'_1, G'_2) is a positive instance of* SEFE. *Further, $G'_{1\cap 2}$ is a tree and all the exclusive edges of G'_1 and of G'_2 are incident only to leaves of $G'_{1\cap 2}$. Moreover, $G'_{1\cap 2}$ has $O(n)$ vertices.*

Proof. $G_{1\cap 2}$ is a tree, by assumption. When an exclusive edge uv in G_1 (respectively in G_2) such that u is not a leaf of $G_{1\cap 2}$ is subdivided with a vertex u' and edge uu' is added to E_2 (respectively to E_1), an edge is inserted into $G_{1\cap 2}$ connecting an internal vertex of $G_{1\cap 2}$ with a new leaf of $G_{1\cap 2}$, namely u'. Hence, $G_{1\cap 2}$ remains a tree after such a modification and thus $G'_{1\cap 2}$ is a tree. When an exclusive edge uv in G_1 (respectively in G_2) such that u is not a leaf of $G_{1\cap 2}$ is subdivided with a vertex u' and edge (u, u') is added to E_2 (resp. to E_1), the number of incidences between exclusive edges and internal vertices of $G_{1\cap 2}$

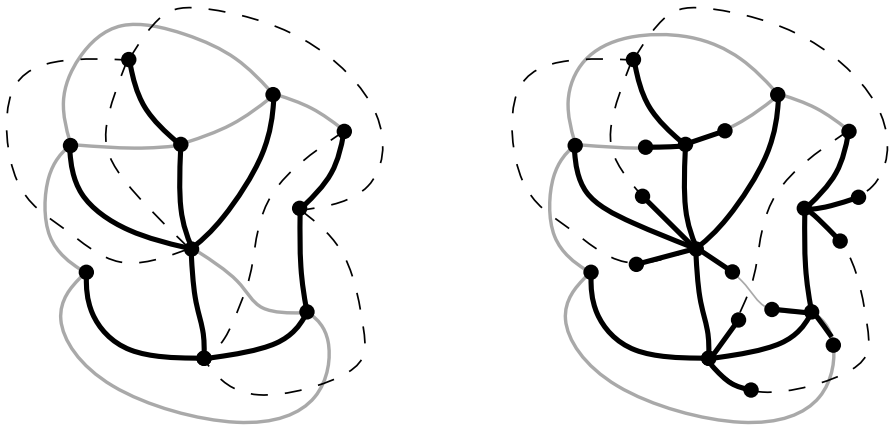


Figure 8.7.: Reduction of SEFE for instance (G_1, G_2) whose intersection graph is a tree to an instance (G'_1, G'_2) whose intersection graph is a tree and all exclusive edges are incident to leaves of the tree. (a) A SEFE Γ of G_1, G_2. (b) A SEFE Γ' of G'_1, G'_2.

decreases by one. Hence, after all such modifications have been performed, all the exclusive edges are incident only to leaves of $G'_{1\cap 2}$. Each exclusive edge is subdivided at most twice. Since the number of edges of $G_{1\setminus 2}$ and $G_{2\setminus 1}$ is $O(n)$, then $G'_{1\cap 2}$ has $O(n)$ vertices. We now prove that (G_1, G_2) is a positive instance of SEFE if and only if (G'_1, G'_2) is a positive instance of SEFE.

First, suppose that a SEFE Γ of (G_1, G_2) exists. Modify Γ to obtain a SEFE Γ' of (G'_1, G'_2) as follows; see Figures 8.7(a) and 8.7(b). When an exclusive edge uv in G_1 (respectively in G_2) such that u is not a leaf of $G_{1\cap 2}$ is subdivided with a vertex u' and edge uu' is added to E_2 (respectively to E_1), insert u' in Γ along the drawing of the edge uv, arbitrarily close to u. Since the drawing of G_1 in Γ is not modified and since the drawing of G_2 in Γ is modified by inserting an arbitrarily small edge incident to a vertex, the resulting drawing is a SEFE of the current graphs and hence Γ' is a SEFE of (G'_1, G'_2).

Second, suppose that a SEFE Γ' of (G'_1, G'_2) exists. A SEFE Γ of (G_1, G_2) can be obtained by drawing each edge uv of G_1 (respectively of G_2) exactly as in Γ'. Observe that uv is subdivided never, once, or twice in G'_1 (respectively in G'_2); then, its drawing in Γ is composed of the concatenation of the one, two, or three curves representing the parts of uv in Γ'. That no two edges of G_1 (respectively of G_2) intersect in the resulting drawing Γ directly descends from the fact that no two edges of G'_1 (respectively of G'_2) intersect in Γ'. Hence Γ is a SEFE of (G_1, G_2). □

In the third step, we transform an instance (G_1, G_2) of SEFE such that $G_{1\cap 2}$ is a tree and all the exclusive edges of G_1 and of G_2 are incident only to leaves of $G_{1\cap 2}$ into an equivalent instance of PARTITIONED T-COHERENT 2-PAGE BOOK EMBEDDING.

The input of PARTITIONED T-COHERENT 2-PAGE BOOK EMBEDDING consists of the graph G composed of all the vertices that are leaves of $G_{1\cap 2}$, of all the exclusive edges $E_{1\setminus 2}$ of $G_{1\setminus 2}$, and of all the exclusive edges $E_{2\setminus 1}$ of $G_{2\setminus 1}$. The partition of the edges of G is $(E_{1\setminus 2}, E_{2\setminus 1})$. Finally, tree T is $G_{1\cap 2}$. We have the following.

Lemma 8.13. *The pair (G_1, G_2) is a positive instance of* SEFE *iff $(G, E_{1\setminus 2}, E_{2\setminus 1}, T)$ is a positive instance of* PARTITIONED T-COHERENT 2-PAGE BOOK EMBEDDING.

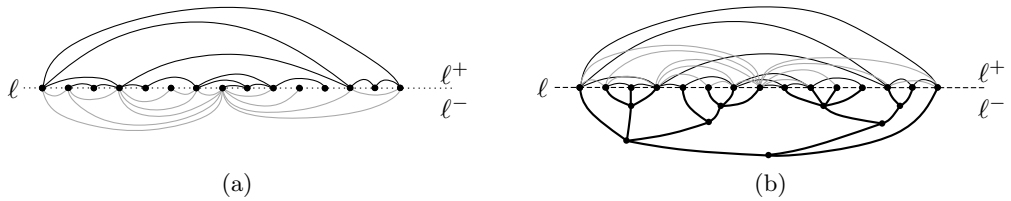


Figure 8.8.: (a) A Partitioned T-coherent 2-page book embedding of $(G, E_{1\setminus 2}, E_{2\setminus 1}, T)$. (b) The SEFE of G_1, G_2 obtained from the book embedding of $(G, E_{1\setminus 2}, E_{2\setminus 1}, T)$.

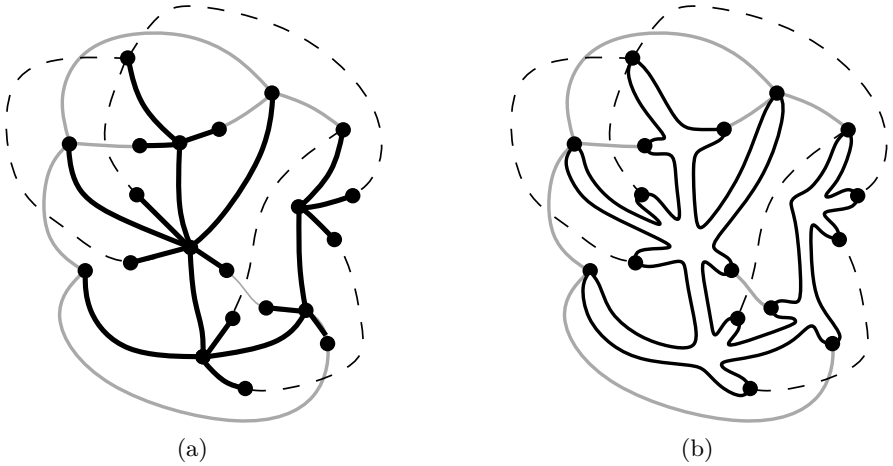


Figure 8.9.: (a) A SEFE Γ of G_1, G_2. (b) Euler Tour $\mathcal{E}$ of $G_{1\cap 2}$ and exclusive edges.

Proof. First, suppose that $(G, E_{1\setminus 2}, E_{2\setminus 1}, T)$ is a positive instance of PARTITIONED T-COHERENT 2-PAGE BOOK EMBEDDING; see Figure 8.8. An ordering of the vertices of G along a line ℓ exists such that the edges in $E_{1\setminus 2}$ are drawn on one side ℓ^+ of ℓ, the edges in $E_{2\setminus 1}$ are drawn on the other side ℓ^- of ℓ, no two edges in $E_{1\setminus 2}$ cross, and no two edges in $E_{2\setminus 1}$ cross. Move all the edges in $E_{2\setminus 1}$ to ℓ^+. Since such edges do not cross in ℓ^- and since the ordering of the vertices of G is not modified, the edges in $E_{2\setminus 1}$ still do not cross. Finally, construct a planar drawing of $G_{1\cap 2}$ in ℓ^-. This can always be done since, for each internal vertex t of $G_{1\cap 2}$, the vertices in the subtree of $G_{1\cap 2}$ rooted at t appear consecutively on ℓ. The resulting drawing is hence a SEFE of (G_1, G_2).

Second, suppose that (G_1, G_2) is a positive instance of SEFE. Consider any SEFE Γ of (G_1, G_2) and consider a planar Euler Tour $\mathcal{E}$ of $G_{1\cap 2}$; see Figures 8.9(a) and 8.9(b). Construct a planar drawing of $\mathcal{E}$ in Γ as follows. Each edge of $\mathcal{E}$ is drawn arbitrarily close to the corresponding edge in $G_{1\cap 2}$. Each endpoint t of an edge of $\mathcal{E}$ that is a leaf in $G_{1\cap 2}$ is drawn at the same point where it is drawn in Γ. Each endpoint t of an edge of $\mathcal{E}$ that is not a leaf in $G_{1\cap 2}$ and that has two adjacent edges tt_1 and tt_2 in $\mathcal{E}$ (observe that $t_1 \neq t_2$ as t is an internal vertex of $G_{1\cap 2}$) is drawn arbitrarily close to the point where t is drawn in Γ, in the region "between" edges tt_1 and tt_2. Clearly, the resulting drawing of $\mathcal{E}$ is planar.

Further, all the leaf vertices of $G_{1\cap 2}$ are drawn at the same point in Γ and in the drawing of $\mathcal{E}$. Moreover, all the exclusive edges of $G_{1\setminus 2}$ and all the exclusive edges of $G_{2\setminus 1}$ lie entirely outside $\mathcal{E}$, except for their endpoints. Remove all the internal vertices and all the edges of $G_{1\cap 2}$ from the drawing. Move all the edges of $G_{2\setminus 1}$ inside $\mathcal{E}$. The resulting drawing is a Partitioned T-coherent 2-page book embedding of $(G, E_{1\setminus 2}, E_{2\setminus 1}, T)$. Namely,

all the edges in $E_{1\setminus 2}$ are on one side of $\mathcal{E}$ and all the edges in $E_{2\setminus 1}$ are on the other side of $\mathcal{E}$. No two edges in $E_{1\setminus 2}$ cross as they do not cross in Γ, and analogously for edges in $E_{2\setminus 1}$. Finally, all the leaf vertices in a subtree of $G_{1\cap 2}$ rooted at an internal vertex t of $G_{1\cap 2}$ appear consecutively in $\mathcal{E}$, as the drawing of $G_{1\cap 2}$ in Γ is planar. □

Given an instance (G, E_1, E_2, T) of Partitioned T-coherent 2-page book embedding, it is possible to construct an equivalent instance of Sefe as follows. Let G_1 be the graph whose vertex set is composed of the vertices of G and of the internal vertices of T, and whose edge set is composed of the edges of E_1 and of the edges of T. Analogously, let G_2 be the graph whose vertex set is composed of the vertices of G and of the internal vertices of T, and whose edge set is composed of the edges of E_2 and of the edges of T. Analogous to Lemma 8.13, we can prove the following.

Lemma 8.14. *(G, E_1, E_2, T) is a positive instance of* Partitioned T-coherent 2-page book embedding *if and only if (G_1, G_2) is a positive instance of* Sefe.

Since both reductions can easily be performed in linear time we obtain the following.

Theorem 8.3. Partitioned T-coherent 2-page book embedding *and* Sefe *for two graphs with connected intersection graph have the same time complexity.*

The problem Partitioned T-coherent 2-page book embedding has been recently studied by Hong and Nagamochi [HN09] when T is a star, that is, the graph has the edges partitioned into two pages as part of the input, but there is no constraint on the order of the vertices in the required book embedding. In this case, Hong and Nagamochi proved that the problem is $O(n)$-time solvable [HN09]. While their motivation was a connection to the c-planarity problem, Lemmas 8.11–8.13 together with Hong and Nagamochi's result imply that deciding whether a Sefe exists for two n-vertex graphs whose intersection graph is a star is a linear-time solvable problem.

Theorem 8.4. *The* Sefe *problem for two n-vertex graphs G_1 and G_2 is solvable in $O(n)$ time if the intersection graph $G_{1\cap 2}$ of G_1 and G_2 is a star.*

8.5. Concluding Remarks

In this chapter we have shown new results on the time complexity of the problem of deciding whether two planar graphs admit a Sefe.

First, we have shown that the Sefe problem can be solved in polynomial time if the intersection graph $G_{1\cap 2}$ of the input graphs G_1 and G_2 is biconnected. Using dynamic programming, we further refined our algorithm and gave a linear-time implementation. More generally, with similar techniques it is also possible to solve in polynomial time the Sefe problem if $G_{1\cap 2}$ consists of one biconnected component plus a set of isolated vertices.

Second, we have shown that when $G_{1\cap 2}$ is connected the Sefe problem can be equivalently stated as a 2-page book embedding problem with edges assigned to the pages and with hierarchical constraints. Hence, pursuing an NP-hardness proof for such a book embedding problem is a possible direction for trying to prove NP-hardness for the Sefe problem. Essentially, the 2-page book embedding problem with hierarchical constraints can be seen as a book embedding problem with three pages: one for each of the two edge sets and one into which the tree is embedded. Hence our results show that Sefe for graphs

whose intersection is connected, corresponds to a book embedding problem with three pages, where two pages contain a matching and one contains a tree.

The problems treated in this chapter involve non-planar drawings of graphs, where crossings are restricted to occur only between certain edges. Although the constraints are still very strict, it can be seen that the arguments to handle such drawings are more complicated than those for handling planar embeddings. One reason is that such drawings are more difficult to describe combinatorially. The combinatorial description for such drawings by Jünger and Schulz, who characterize SEFEs as pairs of planar embeddings whose restrictions to the intersection graph coincide, are more difficult to handle than ordinary planar embeddings. A considerable amount of work went into trying to lift as many statements and arguments of this chapter as possible from the level of drawings to the more abstract level of embeddings, which often allow for simpler and cleaner arguments and offer a broader theoretical toolbox. For some proofs, such as Lemma 8.11, we still had to go down at the level of drawings and really manipulate individual curves in the drawing to achieve the results. In our opinion, this shows that there is still a severe lack in understanding of the combinatorial properties of simultaneous embeddings, which in turn seems to be a prerequisite for giving efficient algorithms.

Open Problems. Clearly, the most important question, the complexity of the SEFE problem, still remains open. In order to understand more about the combinatorial properties of simultaneous embeddings, we suggest two more restricted problems.

First, the following generalization of the SEFE problem with $G_{1\cap 2}$ biconnected seems worth to be tackled. What is the time complexity of computing a SEFE when $G_{1\cap 2}$ is *2-edge connected*?

Second, to further investigate the book-embedding problem motivated by SEFE for graphs whose intersection is connected, it may be useful to consider an even more restricted version, for example by replacing the tree by a third matching. This leads to the following problem, which we call k-PAGEBOOKMATCHING. Given a set V of n vertices and k (perfect) matchings $E_1, \ldots, E_k$ on V, does there exist a total ordering $\prec$ on V such that no two edges $ab, cd \in E_i$ with $a \prec c \prec b \prec d$ exist for $i = 1, \ldots, k$, that is, each matching can be drawn in a planar way on its own page with the ordering of the vertices along the spine fixed to $\prec$. For $k = 1, 2$ this problem is in $\mathcal{P}$ by the result of Hong and Nagamochi [HN09], even if E_1 and E_2 are arbitrary edge sets, not just matchings. The complexity status of this problem for $k \geq 3$ is unclear. Hence this seems to be a good starting point for further research. The main questions are the following. What is the complexity of k-PAGEBOOKMATCHING? What is the complexity of the problem for fixed values of k, and if this turns out to be tractable, is it fixed-parameter tractable with respect to parameter k? That is, is there an algorithm with running time $O(f(k)n^c)$ where c is a constant and f is an arbitrary function, depending only on k?

Chapter 9

Orthogonal Graph Drawing with Flexibility Constraints

Until now, we have studied two different embedding styles of planar graphs, namely geometric embeddings, where edges are drawn as straight lines and topological drawings where curves are represented by arbitrary Jordan curves. In this chapter, we study another drawing style, where edges are represented by curves that are piecewise axis-parallel, so-called *orthogonal drawings*. Such a drawing can also be seen as an embedding of a graph into a grid.

Traditionally, the quality of orthogonal planar drawings is quantified by either the total number of bends, or the maximum number of bends per edge. However, this neglects that in typical applications, edges have varying importance. In this chapter, we investigate an approach that allows to specify the maximum number of bends for each edge individually, depending on its importance.

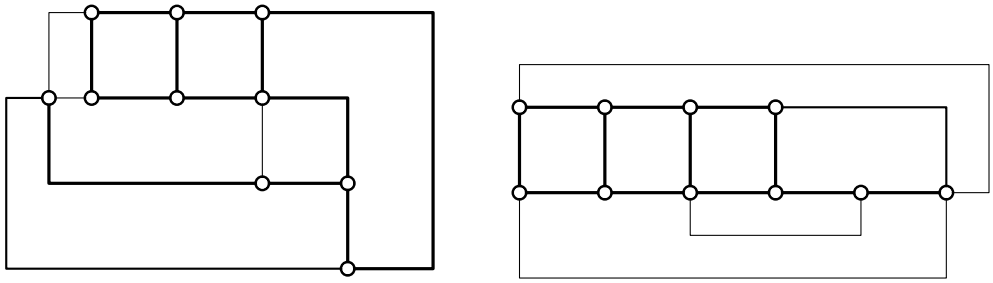
Given a planar graph $G = (V, E)$ on n vertices with maximum degree 4 and a function $\text{flex} : E \longrightarrow \mathbb{N}_0$ that assigns a *flexibility* to each edge, does G admit a planar embedding on the grid such that each edge e has at most $\text{flex}(e)$ bends? Note that in our setting the combinatorial embedding of G is not fixed.

We give an algorithm with running-time $O(n^2)$ for this problem when the flexibility of each edge is positive. This includes, as a special case, the problem of deciding whether G admits a drawing with at most one bend per edge.

The chapter is based on joint work with Thomas Bläsius, Marcus Krug, and Dorothea Wagner [BKRW11].

9.1. Introduction

Orthogonal graph drawing is one of the most important techniques for the human-readable visualization of complex data. Its æsthetic appeal derives from its simplicity and straightforwardness. Since edges are required to be straight orthogonal lines—which automatically yields good angular resolution and short links—the human eye may easily adapt to the flow of an edge. The readability of orthogonal drawings can be further enhanced in the absence of crossings, that is, if the underlying data exhibits planar structure. Unfortunately, not all planar graphs have an orthogonal drawing in which each edge may be represented by a straight horizontal or vertical line. In order to be able to visualize all planar graphs with maximum degree 4, nonetheless, we allow edges to have bends. Since bends obfuscate the



(a) (b)

Figure 9.1.: Two orthogonal drawings of the same graph. The thickness of edges indicates their importance. Although, the drawing in (a) has both fewer bends and fewer bends per edge, drawing (b) is much clearer since important edges have fewer bends.

readability of orthogonal drawings, however, we are interested in minimizing the number of bends on the edges. Previous approaches to orthogonal graph drawing in the presence of bends focus on either the minimization of the maximum number of bends per edge or the total number of bends in the drawing.

In typical applications, however, edges have varying importance for the readability depending on their semantic and their importance for the application. Thus, it is convenient to allow some edges to have more bends than others; see Figure 9.1 for an example.

We consider the following orthogonal graph drawing problem, which we call FlexDraw. Given a *4-planar graph* G, that is, G is planar and has maximum degree 4, and for each edge e a non-negative integer $\text{flex}(e)$, its *flexibility*, does G admit a planar embedding on the grid such that each edge e has at most $\text{flex}(e)$ bends? Such a drawing of G on the grid is called a *flex-drawing*. For a graph with $\text{flex}(e) > 0$ for each edge e in G we shortly say that G has *positive flexibility*.

Related work. The problem we consider generalizes a well-studied problem in orthogonal graph drawing, namely the problem of deciding whether a given graph is β-embeddable for some non-negative integer β. A 4-planar graph is *β-embeddable* if it admits an embedding on the grid with at most β bends per edge.

Garg and Tamassia [GT01] show that it is NP-hard to decide 0-embeddability. The reduction crucially relies on the construction of graphs with rigid embeddings. Later, we show that this is impossible if we allow at least one bend per edge. This is a key observation, which forms the basis for an efficient algorithm for recognizing 1-embeddable graphs. For special cases, namely planar graphs with maximum degree 3 and series-parallel graphs, Di Battista et al. [DLV98] gave an efficient algorithm that minimizes the total number of bends and hence solves 0-embeddability. On the other hand, Biedl and Kant [BK94] show that every 4-planar graph admits a drawing with at most two bends per edge, with the only exception of the octahedron, which requires an edge with three bends. They present a linear-time algorithm for connected—and not necessarily planar—graphs with maximum degree 4 that computes an embedding on a grid of size $n \times n$ with at most two bends per edge and at most $2n + 2$ bends in total. The resulting embeddings are plane for planar graphs. Similar results are obtained by Liu et al. [LMS98].

Liu et al. [LMPS92] claim to have found a characterization of the planar graphs with minimum degree 3 and maximum degree 4 that admit an orthogonal embedding with at most one bend per edge. They also claim that this characterization can be tested in polynomial time. Unfortunately, their paper does not include any proofs and to the best of

our knowledge a proof of these results has not been published. Morgana et al. [MdMS04] characterize the class of *plane graphs* (that is, planar graphs with a given embedding) that admit a 1-bend embedding on the grid by forbidden configurations. They also present a quadratic-time algorithm that either detects a forbidden configuration or computes a 1-bend embedding.

Similar questions have also been considered for the case that edges are not required to be orthogonal. Kaufmann and Wiese [KW02] show that every 4-connected plane graph can be embedded with at most one bend per edge on any given point set and that general plane graphs can be embedded with at most two bends per edge in this drawing style.

If the combinatorial embedding of a 4-planar graph is given, Tamassia's flow network can be used to minimize the total number of bends [Tam87]. Note that this approach may yield drawings with a linear number of bends for some of the edges. Given a combinatorial embedding that admits a 1-bend embedding, however, the flow network can be modified in a straightforward manner to minimize the total number of bends using at most one bend per edge.

The problem we consider involves considering all embeddings of a planar graph. Many problems of this sort are NP-hard. For instance, 0-embeddability is NP-hard [GT01], even though it can be decided efficiently if we are given an embedding by minimizing the total number of bends. On the other hand, some interesting problems have been shown to be efficiently solvable, even if they involve optimization over all embeddings. Gutwenger et al. [GMW01] show how to efficiently compute an embedding with the minimum number of crossings for a planar graph with an additionally added edge. In a follow-up paper Chimani et al. [CGMW09] show how to achieve the same for apex graphs, that is, planar graphs that have been augmented by an additional vertex and an arbitrary number of edges incident to this vertex. Also the results presented in Chapters 6 and 8 fall into this category.

Contribution and Outline. In this chapter we give an efficient algorithm with running time $O(n^2)$ that solves FlexDraw for graphs with positive flexibility. Since FlexDraw contains the problem of 1-embeddability as a special case, this closes the complexity gap between the NP-hardness result for 0-embeddability by Garg and Tamassia [GT01] and the efficient algorithm for computing 2-embeddings by Biedl and Kant [BK94]. Note that once we have found a feasible planar embedding, a corresponding drawing can be computed in $O(n^2)$ time [MdMS04].

We present some preliminaries in Section 9.2. In Section 9.3 we study orthogonal flex-drawings of graphs with a fixed embedding and introduce the maximum rotation of a graph as a measure of how "flexible" it is. In Section 9.4 we show that replacing certain subgraphs with graphs that behave similarly with respect to flexibility does not change the maximum rotation. Based on this fact and the SPQR-tree we give an algorithm that solves FlexDraw for biconnected 4-planar graphs with positive flexibility. In Section 9.5 we improve the running time of our algorithm to $O(n^2)$. We extend our algorithm to arbitrary 4-planar graphs with positive flexibility in Section 9.6 and discuss some extensions and possible directions for future work in Section 9.8.

9.2. Preliminaries

In this section we introduce notations and preliminaries that are essential throughout this chapter. These are mainly the orthogonal representation as a combinatorial description of orthogonal drawings and Tamassia's flow network as an algorithmic tool for handling such descriptions in the context of a fixed embedding.

SPQR-tree and st-graphs. A *weak st-graph* is a 4-planar graph $G = (V, E)$ with two designated vertices s and t such that the graph $G + st$ is planar and has maximum degree 4. An *st-graph* is a weak st-graph such that $G+st$ is biconnected. An orthogonal representation $\mathcal{R}$ of a (weak) st-graph with positive flexibility is *valid* if each edge e has at most $\text{flex}(e)$ bends and s and t are embedded on the outer face. A valid orthogonal representation of a (weak) st-graph is *tight* if all angles at s and t in inner faces are 90°.

We distinguish st-graphs with $\deg(s), \deg(t) \leq 2$ by the degrees of s and t. An st-graph is of Type (1,1) if $\deg(s) = \deg(t) = 1$, it is of Type (1,2) if one of them has degree 1 and the other one has degree 2 and it is of Type (2,2) if $\deg(s) = \deg(t) = 2$.

Our algorithm for solving FlexDraw makes use of the SPQR-tree to handle all possible embeddings. Since orthogonal drawings require special handling of the external face, we use the rooted version of the SPQR-tree, which represents embeddings where the *reference edge*, that is the edge at whose Q-node the tree is rooted, is incident to the outer face. Note that the pertinent graph of a node always forms an st-graph with respect to the split pair it shares with its parent and that for non-R-nodes, the pertinent graph is of Type (1,1), (1,2) or (2,2).

Orthogonal representation. The *orthogonal representation* introduced by Tamassia describes orthogonal drawings of plane graphs by listing the faces as sequences of bends [Tam87]. As an advantage the orthogonal representation neglects the lengths of segments. Thus, it is possible to manipulate drawings without the need to worry about the exact geometry. Our orthogonal representation is always normalized, that is, each edge has only bends in one direction; this slightly differs from the notion introduced by Tamassia. This is not a restriction since every orthogonal representation can be normalized.

The orthogonal representation of a plane graph G is defined as a set of lists $\mathcal{R}$ containing a list $\mathcal{R}(f_i)$ for each face f_i of G. For each face f_i the list $\mathcal{R}(f_i)$ is a circular list of *edge descriptions* containing the edges on the boundary of f_i in clockwise order(counter-clockwise if f_i is the external face). Each description $r \in \mathcal{R}(f_i)$ contains the following information: $\text{edge}(r)$ denotes the edge represented by r, $\text{bends}(r)$ is an integer whose absolute value is the number of 90°-bends of $\text{edge}(r)$, where positive numbers represent bends to the right and negative numbers bends to the left. For a given edge description $r \in \mathcal{R}(f_i)$ we denote its successor in $\mathcal{R}(f_i)$ by r' and represent the angle α between $\text{edge}(r)$ and $\text{edge}(r')$ in f_i by their rotation $\text{rot}(r, r') = 2 - \alpha/90°$. Every edge has exactly two edge descriptions, if r is one of them, the other is denoted by $\bar{r}$. Since each face forms a rectilinear polygon, every orthogonal representation $\mathcal{R}$ of an orthogonal drawing has the following three properties.

(I) Each edge description r is consistent with $\bar{r}$, that is, $\text{bends}(\bar{r}) = -\text{bends}(r)$.

(II) The interior bends of any face f sum up to 4 and the exterior bends to -4:

$$\sum_{r \in \mathcal{R}(f)} (\text{bends}(r) + \text{rot}(r, r')) = \begin{cases} -4, & \text{if } f \text{ is the external face,} \\ +4, & \text{if } f \text{ is an internal face.} \end{cases}$$

(III) The angles around every node sum up to 360°.

Given an orthogonal representation $\mathcal{R}$ of a graph, a corresponding orthogonal drawing can be computed efficiently [Tam87]. Hence, it is sufficient to work with orthogonal representations. An orthogonal representation is *valid* for a given flexibility function flex if $|\operatorname{bends}(r)| \leq \operatorname{flex}(\operatorname{edge}(r))$ for each edge description r.

For a planar graph $G = (V, E)$ with orthogonal representation $\mathcal{R}$ and two vertices s and t on the outer face f_1, we denote by $\pi_{\mathcal{R}}(s, t)$ the unique simple path in $\mathcal{R}(f_1)$ that connects s and t in counter-clockwise direction. Such a path $\pi = \pi(s, t)$ consists of consecutive edge descriptions $r_1, \ldots, r_k$. We define the *rotation* of π as

$$\operatorname{rot}_{\mathcal{R}}(\pi) = \sum_{i=1}^{k} \operatorname{bends}(r_i) + \sum_{i=1}^{k-1} \operatorname{rot}(r_i, r_{i+1}).$$

Moreover, for the vertex s we denote by $\operatorname{rot}_{\mathcal{R}}(s)$ the rotation value of the angle between $\pi_{\mathcal{R}}(s, t)$ and $\pi_{\mathcal{R}}(t, s)$ at s. We define $\operatorname{rot}_{\mathcal{R}}(t)$ analogously. Note that, for a single edge description r we have $\operatorname{rot}_{\mathcal{R}}(r) = \operatorname{bends}(r)$. If it is clear from the context which orthogonal representation is meant we omit the indices of π and rot. The concept of rotation is similar to the spirality defined by Di Battista et al. [DLV98].

The value $\operatorname{rot}(\pi(s, t))$ describes the shape of the path $\pi(s, t)$ in the orthogonal representation in terms of the angle between its start- and its endpoint. Fixing the rotation of $\pi(s, t)$, $\pi(t, s)$ and the outer angles at s and t in a sense determines the shape of the outer face. In Section 9.4, we will exploit this by replacing certain subgraphs of G with simpler graphs whose outer faces have the same shapes.

Flows and Tamassia's flow network. A *flow network* is a tuple $N = (V, A, \ell, u, q)$ where (V, A) is a directed (multi-)graph, $\ell : A \longrightarrow \mathbb{N}_0$ and $u : A \longrightarrow \mathbb{N}_0 \cup \{\infty\}$ are lower and upper bounds for the amount of flow along the arcs in A with $\ell(a) \leq u(a)$ for all $a \in A$. Finally, $q : V \longrightarrow \mathbb{Z}$ defines a demand for each vertex. Note that demands can be positive or negative.

A *flow* is a function $\phi : A \longrightarrow \mathbb{N}_0$ that maps a certain amount of flow to each arc such that $\ell(a) \leq \phi(a) \leq u(a)$ holds for all arcs $a \in A$. A flow ϕ is *feasible*, if in addition the difference of incoming and outgoing flow at each vertex equals its demand, that is,

$$q(v) = \sum_{(u,v)\in A} \phi(u, v) - \sum_{(v,u)\in A} \phi(v, u) \quad \text{for all } v \in V.$$

The defect of a node v with respect to a flow ϕ is defined as

$$\operatorname{def}_{\phi}(v) = \left| \sum_{(u,v)\in A} \phi(u, v) - \sum_{(v,u)\in A} \phi(v, u) - q(v) \right| .$$

The *defect of a flow* ϕ is defined as $\operatorname{def}(\phi) = \sum_{v\in V} \operatorname{def}_{\phi}(v)$. Clearly, a flow has defect 0 if and only if it is feasible. It is not hard to see that given a flow ϕ, a flow ϕ' with minimum defect can be computed in time $O(\operatorname{def}(\phi)|N|)$ using the algorithm of Ford and Fulkerson [FF56].

Let $G = (V, E)$ be a 4-planar graph together with a planar embedding $\mathcal{E}$ and let $F = \{f_1, \ldots, f_k\}$ be the faces of G with respect to embedding $\mathcal{E}$, where f_1 is the outer face. Further let n_i be the number of vertices that are incident to f_i.

Tamassia's flow network consists of nodes $V \cup F$ with $q(v) = -4$ for all $v \in V$, $q(f_i) = 2n_i + 4$ for $i \geq 2$ and $q(f_1) = 2n_0 - 4$. The flow network contains the following

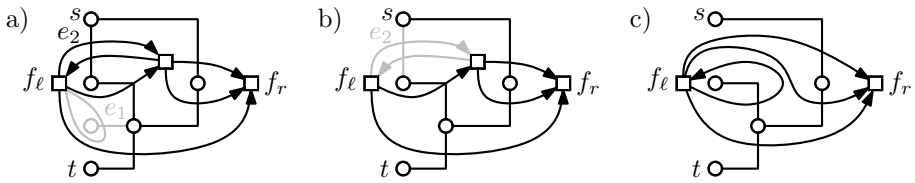


Figure 9.2.: An st-graph with flexibility 1 for all edges with $\text{rot}(\pi(s,t)) = 1$ and its flex graph $G^\times$ (a), after removal of bridge e_1 (b), and removal of edge e_2 (c).

arcs. For each node v let F_v be the faces incident to v. Then, for each face $f \in F_v$ there is an arc a from v to f with $\ell(a) = 1$ and $u(a) = 4$. Further, for each edge e of G with incident faces g and h there is an arc a_1 from g to h and an arc a_2 from h to g with $\ell(a_1) = \ell(a_2) = 0$ and $u(a_1) = u(a_2) = \infty$. The demands and capacities essentially represent the distribution of 90°-angles around vertices and faces. Tamassia showed that there is a bijection between feasible flows in the network and orthogonal representations of G with embedding $\mathcal{E}$ [Tam87]. FlexDraw with fixed embedding can easily be handled by setting the upper bound of the arcs stemming from an edge e to $\text{flex}(e)$ for all edges $e \in E$.

Let e be an edge of G with incident faces g and h and let a_1 and a_2 be the two arcs stemming from e such that a_1 is directed from g to h and a_2 is directed from h to g. Note that for a flow ϕ, by eliminating cycles with positive flow, we may assume that either $\phi(a_1) = 0$ or $\phi(a_2) = 0$ holds. For ease of notation, we therefore identify these two arcs with e and write $\phi_e(g,h)$ for the amount of flow from g to h via the arcs stemming from e. Note that this value can be negative if there is flow from h to g and we have $\phi_e(g,h) = -\phi_e(h,g)$.

Proof outline. We start out with an observation. Let G be a 4-planar graph with positive flexibility and let $\{s,t\}$ be a split pair of G that splits G into two subgraphs G_1 and G_2, and let e_{ref} be an edge of G_1. Let ρ be the maximum rotation of $\pi(s,t)$ over all embeddings of G_2 where s and t are on the outer face.

If G_2 is of Type (1,1) then obviously the following holds. If G admits a valid orthogonal drawing with the given flexibility such that e_{ref} is embedded on the outer face then also the graph G' where G_2 is replaced by the single edge st with flexibility ρ admits such a drawing. Graphs of Type (1,2) and (2,2) allow for similar substitutions.

Thus, we can substitute st-graphs of each type with a small gadget graph to obtain a new graph G' with the property that G' has a valid drawing if G has one. We show that the converse is also true, that is, if the graph G' admits such an embedding then also G does. We then exploit this characterization algorithmically, using the SPQR-tree of G to successively replace subgraphs of G by simpler graphs.

9.3. The Maximum Rotation with a Fixed Embedding

The goal of this section is to derive a description of the valid orthogonal representations of a given (weak) st-graph with positive flexibility and a fixed embedding. Namely, we prove that the values that can be obtained for $\text{rot}(\pi(s,t))$ form an interval for these graphs. We show that if there exists a valid orthogonal representation $\mathcal{R}$ with $\text{rot}_{\mathcal{R}}(\pi(s,t)) \geq 0$ then there exists an orthogonal representation $\mathcal{R}'$ with $\text{rot}_{\mathcal{R}'}(\pi(s,t)) = \text{rot}_{\mathcal{R}}(\pi(s,t)) - 1$, which can be obtained from $\mathcal{R}$ by only altering the number of bends on certain edges.

To model the possible changes of an orthogonal representation $\mathcal{R}$ of a (weak) st-graph G

that can be performed by only changing the number of bends on edges we introduce the *flex graph* $G^\times$ of G with respect to $\mathcal{R}$, which is based on the bidirected dual graph of G. Thus, the flex graph is a directed multigraph; see Figure 9.2a for an illustration. We start out by adding to G the edge st and embed it into the outer face of G, thus splitting the outer face into two faces f_ℓ and f_r, where f_ℓ is bounded by $\pi(s,t)$ and the new edge $\{s,t\}$ and f_r is bounded by $\pi(t,s)$ and $\{s,t\}$. We denote this graph by $\bar{G}$ and its dual graph by $\bar{G}^*$. We set $V^\times = V(\bar{G}^*)$ and we define $E^\times$ as follows. For each edge e of G denote its incident faces in $\bar{G}$ by f_u and f_v and let r_u and r_v be the edge descriptions of e in $\mathcal{R}(f_u)$ and $\mathcal{R}(f_v)$, respectively. We add the edge (f_u, f_v) if $-\operatorname{flex}(e) < \operatorname{bends}(r_u)$ and, analogously, we add (f_v, f_u) if $-\operatorname{flex}(e) < \operatorname{bends}(r_v)$. Consider an edge (f_u, f_v) of $G^\times$ and let r_u and r_v be the edge descriptions of the corresponding edge e in G. The fact that $(f_u, f_v) \in E^\times$ indicates that it is possible to decrease $\operatorname{bends}(r_u)$ (and thus increase $\operatorname{bends}(r_v)$) by at least 1 without violating the flexibility of e.

Assume that there exists a simple directed path from f_ℓ to f_r in $G^\times$. Let $f_\ell = f_1, f_2, \ldots, f_k = f_r$ be this path. We construct a new orthogonal representation $\mathcal{R}'$ from $\mathcal{R}$ as follows. For each edge $f_i f_{i+1}$, $i = 1, \ldots, k-1$, let e_i be the corresponding edge of G and let $r_i \in \mathcal{R}(f_i), \bar{r}_i \in \mathcal{R}(f_{i+1})$ be its edge descriptions. We obtain $\mathcal{R}'$ from $\mathcal{R}$ by decreasing $\operatorname{bends}(r_i)$ by 1 and increasing $\operatorname{bends}(\bar{r}_i)$ by 1 for $i = 1, \ldots, k-1$. First, it is clear that $\mathcal{R}'$ satisfies Properties I and III since we increase and decrease the number of bends consistently and we do not change any angles at vertices. Property II holds since each face of G has either none of its edge descriptions changed or exactly one of them is increased by 1 and exactly one of them is decreased by 1. Moreover, since the path starts at f_ℓ and ends at f_r we have that $\operatorname{rot}_{\mathcal{R}'}(\pi(s,t)) = \operatorname{rot}_{\mathcal{R}}(\pi(s,t)) - 1$. We now show that such a path exists if $\operatorname{rot}(\pi(s,t)) \geq 0$.

Lemma 9.1. *Let G be a weak st-graph with positive flexibility and let $\mathcal{R}$ be a valid orthogonal representation of G with $\operatorname{rot}_{\mathcal{R}}(\pi(s,t)) \geq 0$. Then the flex graph $G^\times$ contains a directed path from f_ℓ to f_r.*

Proof. Assume that G is a minimal counter example such that $G^\times$ does not contain such a path. First, we show that in $G^\times$ there exists at least one edge starting from f_ℓ. Let $\pi(s,t)$ be composed of the edge descriptions $r_1, \ldots, r_k$ in $\mathcal{R}(f)$, where f is the outer face of G. Then, by assumption we have $\operatorname{rot}(\pi(s,t)) = \sum_{i=1}^{k} \operatorname{bends}(r_i) + \sum_{i=1}^{k-1} \operatorname{rot}(r_i, r_{i+1}) \geq 0$. Since $\operatorname{rot}(r_i, r_{i+1}) \leq 1$ for $i = 1, \ldots, k-1$ we have that $\sum_{i=1}^{k} \operatorname{bends}(r_i) \geq -k+1$ and hence there is at least one r_j with $\operatorname{bends}(r_j) \geq 0$. Hence, $G^\times$ contains an edge corresponding to $\operatorname{edge}(r_j)$ that starts at f_ℓ. This shows that there always exists an edge (f_ℓ, f_u) in $G^\times$. We distinguish three types of edges (f_ℓ, f_u). If $f_u = f_r$, then (f_ℓ, f_u) is the desired path.

If $f_u = f_\ell$, the corresponding edge e of G is a bridge whose removal does not disconnect s and t; see Figure 9.2b. Then let H be the connected component of $G - e$ containing s and t and let $\mathcal{S}$ be the restriction of $\mathcal{R}$ to H. For the outer face of H we have that $\operatorname{rot}_{\mathcal{S}}(\pi(s,t)) + \operatorname{rot}_{\mathcal{S}}(s) + \operatorname{rot}_{\mathcal{S}}(\pi(t,s)) + \operatorname{rot}_{\mathcal{S}}(t) = -4$. Since $\pi_{\mathcal{R}}(t,s) = \pi_{\mathcal{S}}(t,s)$ we have that $\operatorname{rot}_{\mathcal{S}}(\pi(t,s)) = \operatorname{rot}_{\mathcal{R}}(\pi(t,s))$. Moreover, since we only remove edges the angles at s and t (and thus their rotations) do not decrease, that is, we have $\operatorname{rot}_{\mathcal{S}}(t) \leq \operatorname{rot}_{\mathcal{R}}(t)$ and $\operatorname{rot}_{\mathcal{S}}(s) \leq \operatorname{rot}_{\mathcal{R}}(s)$. Hence, we have that $\operatorname{rot}_{\mathcal{S}}(\pi(s,t)) \geq -4 - \operatorname{rot}_{\mathcal{R}}(\pi(t,s)) - \operatorname{rot}_{\mathcal{R}}(s) - \operatorname{rot}_{\mathcal{R}}(t) = \operatorname{rot}_{\mathcal{R}}(\pi(s,t)) \geq 0$. Since H has fewer edges than G it is not a counter example and its flex graph $H^\times$ contains a path from f_ℓ to f_r. Since $H^\times$ is a subgraph of $G^\times$ this contradicts the assumption that G is a counter example.

Otherwise, f_u is an internal face of G; see Figure 9.2c. Let e be the corresponding edge of G. Let $H := G - e$ and let $\mathcal{S}$ be the orthogonal representation $\mathcal{R}$ restricted to

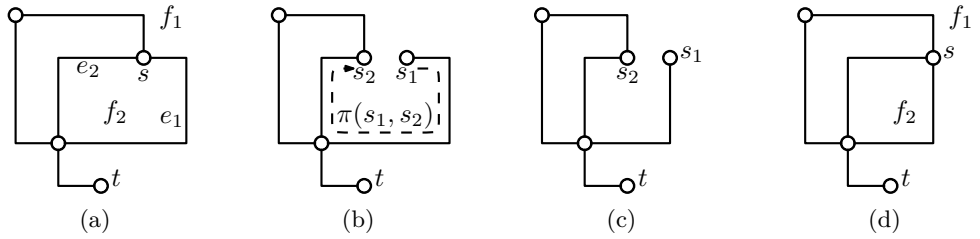


Figure 9.3.: Orthogonal representation that is not tight since s has an angle of 180° in f_2 (a). Splitting s into s_1 and s_2 yields the path $\pi(s_1, s_2)$ with rotation at least 4 (b), hence the rotation can be reduced (c). Merging s_1 and s_2 back into s yields a tight orthogonal representation (d).

H. Note that the flex graph of $H^\times$ of H can be obtained from $G^\times$ by removing all edges between f_ℓ and f_u and merging f_ℓ and f_u into a single node f'_ℓ. As above we obtain that $\text{rot}_\mathcal{S}(\pi(s,t)) \geq 0$ and hence in $H^\times$ there exists a path from f'_ℓ to f_r. The corresponding path in $G^\times$ (after undoing the contraction of f_ℓ and f_u) either starts at f_ℓ or at f_u and ends at f_r. In the former case we have found our path, in the latter case the path together with the edge (f_ℓ, f_u) forms the desired path. Again this contradicts the assumption that G is a counter example. □

Recall that a valid orthogonal representation of a (weak) st-graph is tight if the inner angles at s and t are 90°. We show that a valid orthogonal representation can be made tight without decreasing $\text{rot}(\pi(s,t))$. The proof is illustrated in Figure 9.3.

Lemma 9.2. *Let G be a weak st-graph with positive flexibility and let $\mathcal{R}$ be a valid orthogonal representation. Then there exists a valid orthogonal representation $\mathcal{R}'$ of G with the same planar embedding such that $\mathcal{R}'$ is tight,* $\text{rot}_{\mathcal{R}'}(\pi(s,t)) \geq \text{rot}_\mathcal{R}(\pi(s,t))$ *and* $\text{rot}_{\mathcal{R}'}(\pi(t,s)) \geq \text{rot}_\mathcal{R}(\pi(t,s))$.

Proof. Let f_1 be the outer face and assume that f_2 is an inner face incident to s whose inner angle at s is larger than 90°. We show how to decrease this angle by 90° by only changing the number of bends on certain edges. Hence, by applying the described operation iteratively, we can reduce all internal angles at inner faces incident to s and t to 90°.

Let e_1 and e_2 be the two edges incident to s such that e_1 occurs before e_2 when traversing the boundary of f_2 clockwise starting from s. Assume that e_1 is incident to f_1 (the case that only e_2 is incident to f_1 can be treated analogously).

We split s into two vertices s_1 and s_2. We attach e_1 to s_1 and we attach to s_2 the remaining edges incident to s. Let the resulting graph be H and let $\mathcal{S}$ be the orthogonal representation of H induced by $\mathcal{R}$. Since f_2 is an internal face its total rotation in $\mathcal{R}$ is 4 and since the angle at s was at least 180° we have that $\text{rot}_\mathcal{S}(\pi(s_1, s_2)) \geq 4$. By Lemma 9.1 the flex graph $H^\times$ of H contains a simple path that reduces the rotation along $\pi(s_1, s_2)$ by 1. This path either contains an edge stemming from $\pi(s_2, t)$ or an edge of $\pi(t, s_1)$ and hence either increases $\text{rot}_\mathcal{S}(\pi(s_2, t))$ or $\text{rot}_\mathcal{S}(\pi(t, s_1))$ by 1, whereas the other one remains unchanged. We obtain $\mathcal{R}'$ by merging s_1 and s_2 back into s. Since $\text{rot}_\mathcal{S}(\pi(s_1, s_2))$ was decreased we increase the rotation at s in f_2 by 1 without decreasing $\text{rot}_\mathcal{R}(\pi(s,t)) = \text{rot}_\mathcal{R}(\pi(s_2,t))$ or $\text{rot}_\mathcal{R}(\pi(t,s)) = \text{rot}_\mathcal{R}(\pi(t,s_1))$. Note that aside from changing the number of bends on certain edges we did only change angles incident to s. □

Let G be an st-graph with positive flexibility and fixed planar embedding $\mathcal{E}$. Lemma 9.1 shows that the attainable values of $\text{rot}(\pi(s,t))$ for a given st-graph with a fixed embedding

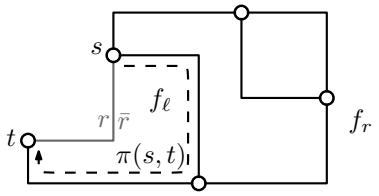


Figure 9.4.: If $\text{rot}(r)$ is maximized, $\text{rot}(\pi(s,t))$ is also maximized and the angles at s and t in f_ℓ are both 90°.

form an interval. Hence, the set of possible rotations can be described by the boundaries of this interval and we define the *maximum rotation* of G with respect to $\mathcal{E}$ as $\text{maxrot}_\mathcal{E}(G) = \max_{\mathcal{R}\in\Omega} \text{rot}_\mathcal{R}(\pi(s,t))$ where Ω contains all valid orthogonal representations of G whose embedding is $\mathcal{E}$.

The following theorem states that indeed the maximum rotation essentially describes the orthogonal representations of st-graphs with fixed embedding and positive flexibility.

Theorem 9.1. *Let G be an st-graph with positive flexibility and fixed embedding $\mathcal{E}$. Then for each $\rho \in \{-1,\dots,\text{maxrot}_\mathcal{E}(G)\}$ there exists a valid and tight orthogonal representation $\mathcal{R}$ of G with planar embedding $\mathcal{E}$ such that $\text{rot}_\mathcal{R}(\pi(s,t)) = \rho$.*

Proof. Let $\rho \in \{-1,\dots,\text{maxrot}_\mathcal{E}(G)\}$. We show how to construct an orthogonal representation $\mathcal{R}$ with $\text{rot}(\pi(s,t)) = \rho$. Let $\mathcal{S}$ be an orthogonal representation of G with embedding $\mathcal{E}$ such that $\text{rot}_\mathcal{S}(\pi(s,t)) = \text{maxrot}_\mathcal{E}(G)$. By Lemma 9.2 we can make $\mathcal{S}$ tight while preserving its embedding and $\text{rot}(\pi(s,t))$. We then apply Lemma 9.1 to reduce $\text{rot}(\pi(s,t))$ to ρ. Note that the representation remains tight as the angles around vertices are not changed by this operation. □

Using a variant of Tamassia's flow network [Tam87] the maximum rotation can be computed efficiently for st-graphs with a fixed embedding.

Theorem 9.2. *Given an st-graph $G = (V,E)$ with fixed embedding $\mathcal{E}$ with s and t on the outer face one can either compute $\text{maxrot}_\mathcal{E}(G)$ or decide that G does not admit a valid orthogonal representation with embedding $\mathcal{E}$ in $O(n^{3/2})$ time.*

Proof. We use the flow network of Tamassia [Tam87] to check whether G admits a valid orthogonal representation with its given embedding. Since this flow network is planar and the in- and out-flow of each sink and source is fixed this can be done in $O(n^{3/2})$ time [MN95].

We add to G the edge st and embed it into the outer face such that we split the outer face of G into two parts f_ℓ and f_r where f_ℓ is bounded by $\pi(s,t)$ and st and f_r is the outer face of $G + st$.

We claim that in a valid orthogonal embedding of $G + st$ that maximizes $\text{rot}(r)$ with its embedding we have that $\text{maxrot}_\mathcal{E}(G) = \text{rot}(r) + 2$ where r is the edge description of st in f_r. Figure 9.4 illustrates this claim and its proof.

The equation $\text{maxrot}_\mathcal{E}(G) \geq \text{rot}(r) + 2$ follows from the fact that in a valid orthogonal embedding of $G + st$ the total rotation in the face f_ℓ is 4. Conversely, by Lemma 9.2 there exists a tight orthogonal representation $\mathcal{R}$ of G with embedding $\mathcal{E}$ such that $\text{rot}(\pi(s,t)) = \text{maxrot}_\mathcal{E}(G)$. Since $\mathcal{R}$ is tight we can attach st in the outer face with $\text{rot}(\pi(s,t)) - 2$ bends. This shows the claim.

Now it remains to show that we can maximize $\text{rot}(r)$ efficiently. We first use the flow network of Tamassia [Tam87] to compute an arbitrary valid orthogonal representation

of $G + st$. To maximize $\text{rot}(r)$ we wish to modify the corresponding flow F in the flow network of Tamassia such that the flow on the edge (f_r, f_ℓ) is maximized while the flow on (f_ℓ, f_r) is 0, which corresponds to maximizing $\text{bends}(r)$. This can be done by computing a maximum flow from f_ℓ to f_r in the residual graph of Tamassia's flow network with respect to F after removing the edges stemming from st. Since this network is planar and the source and the sink lie at the same face a maximum flow can be computed in linear time [HKRS97]. □

9.4. Biconnected Graphs

Until now the planar embedding of our input graph was fixed. Now, we assume that this embedding is variable. Following the approach of the previous section we define the maximum rotation of a (weak) st-graph G as $\text{maxrot}(G) = \max_{\mathcal{E}\in\Psi} \text{maxrot}_{\mathcal{E}}(G)$ where Ψ contains all planar embeddings of G such that s and t are embedded on the outer face.

In this section we show that $\text{maxrot}(G)$ essentially describes all valid orthogonal representations of G in the sense that substituting a subgraph H of G with a different graph H' with $\text{maxrot}(H) = \text{maxrot}(H')$ does not change $\text{maxrot}(G)$. We further use this substitution to give an algorithm that computes maxrot by successively reducing the size of the graph. To handle the different possible planar embeddings we use the SPQR-tree and we substitute subgraphs with small graphs that have only one embedding. We need the following technical lemma.

Lemma 9.3. *Let G be an st-graph with $\deg(s), \deg(t) \leq 2$ and let $\mathcal{R}$ be a tight orthogonal representation of G. Then $\text{rot}(\pi(s,t)) + \text{rot}(\pi(t,s)) = -x$ where x is 0,1 and 2 for graphs of Type (1,1), (1,2) and (2,2), respectively.*

Proof. By property II we have $\text{rot}(\pi(s,t)) + \text{rot}(t) + \text{rot}(\pi(t,s)) + \text{rot}(s) = -4$. If s has degree 1, we have $\text{rot}(s) = -2$. If $\deg(s) = 2$ holds, then s is incident to exactly one inner face and by assumption it has an angle of 90° in this face. Hence, in the outer face there is an angle of 270° and thus $\text{rot}(s) = -1$. As the same analysis holds for t the claim follows. □

The following theorem shows that indeed the maximum rotation describes all possible rotation values of an st-graph.

Theorem 9.3. *Let G be an st-graph with positive flexibility and let ρ be an integer. Then there exists a tight orthogonal representation $\mathcal{R}$ of G with $\text{rot}(\pi(s,t)) = \rho$ if and only if $-\text{maxrot}(G) - x \leq \rho \leq \text{maxrot}(G)$ where x depends on the Type of G and $x = 0, 1, 2$ for Types (1,1), (1,2) and (2,2), respectively.*

Proof. We first show the only if part. Let $\mathcal{R}$ be any embedding of G. By the definition of $\text{maxrot}(G)$ we clearly have that $\text{rot}_{\mathcal{R}}(\pi(s,t)) \leq \text{maxrot}(G)$. By definition we also have that $\text{rot}_{\mathcal{R}}(\pi(t,s)) \leq \text{maxrot}(G)$ (otherwise by mirroring we could obtain an orthogonal representation $\mathcal{R}'$ with $\text{rot}_{\mathcal{R}'}(\pi(s,t)) > \text{maxrot}(G)$) and hence with Lemma 9.3 we obtain $-\text{rot}(\pi(s,t)) - x \leq \text{maxrot}(G)$.

It remains to show that for any given ρ in the range we can find a valid orthogonal representation. If $-1 \leq \rho \leq \text{maxrot}(G)$ we find an orthogonal representation as follows. Let $\mathcal{R}$ be a valid orthogonal embedding of G with $\text{rot}(\pi(s,t)) = \text{maxrot}(G)$. By Lemma 9.2

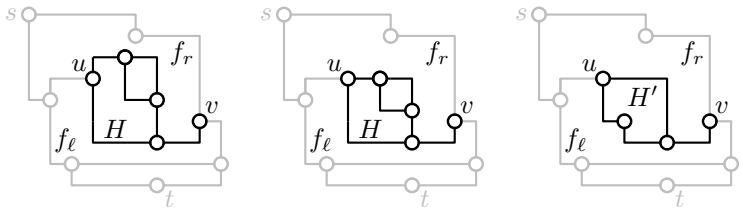


Figure 9.5.: Illustration of Lemma 9.4, st-graph G with split pair $\{u, v\}$ splitting off H (left), replacement of H with a tight orthogonal representation (middle) and replacement of H with a graph H' with $\text{maxrot}(H) = \text{maxrot}(H') = 3$ (right).

we can reduce the inner angles at s and t to 90° without decreasing $\text{rot}(\pi(s,t))$. Then Theorem 9.1 yields the desired orthogonal representation.

If $\rho \leq -2$ holds, by Lemma 9.3 we need to find a valid orthogonal representation $\mathcal{R}$ with $\text{rot}_{\mathcal{R}}(\pi(t,s)) = -\rho - x =: \rho'$. Note that by the definitions of ρ and x we have that $0 \leq \rho' \leq \text{maxrot}(G)$. A valid orthogonal embedding $\mathcal{R}'$ of G with $\text{rot}_{\mathcal{R}'}(\pi(s,t)) = \rho'$ can be found as above. We obtain $\mathcal{R}$ by mirroring $\mathcal{R}'$. □

Note that if s (or t) has degree 1 then its incident edge allows for three different rotations and hence the range of valid rotations contains at least three integers. This observation together with the theorem yields the following.

Corollary 9.1. *Let G be an st-graph with positive flexibility. If G admits a valid drawing then* $\text{maxrot}(G) \geq 1$ *if G is of Type (1,1) or (1,2) and* $\text{maxrot}(G) \geq -1$ *if G is of Type (2,2).*

In particular, Theorem 9.3 shows that an st-graph G with $\deg(s) = \deg(t) = 1$ essentially behaves like a single edge st with flexibility $\text{maxrot}(G)$. The following lemma shows that we can replace any st-graph with $\deg(s), \deg(t) \leq 2$ in a graph G by a different st-graph of the same type and with the same maximum rotation without changing $\text{maxrot}(G)$. Figure 9.5 illustrates the lemma and its proof.

Lemma 9.4. *Let $G = (V, E)$ be an st-graph with positive flexibility and let $\{u, v\}$ be a split pair of G that splits G into two components G^- and H such that G^- contains s and t and H is an st-graph of Type (1,1), Type (1,2) or Type (2,2) (with respect to vertices u and v). Let H' be an st-graph with designated vertices u', v' of the same type as H with* $\text{maxrot}(H') = \text{maxrot}(H)$.

Then G admits a valid orthogonal representation $\mathcal{R}$ with $\text{rot}_{\mathcal{R}}(\pi(s,t)) = \rho$ *if and only if the graph G', which is obtained from G by replacing H with H' admits a valid orthogonal representation $\mathcal{R}'$ with* $\text{rot}_{\mathcal{R}'}(\pi(s,t)) = \rho$.

Proof. Given a valid orthogonal representation $\mathcal{R}$ of G we wish to find a valid orthogonal representation $\mathcal{R}'$ of G' such that $\text{rot}_{\mathcal{R}}(\pi(s,t)) = \text{rot}_{\mathcal{R}'}(\pi(s,t))$. The other direction is symmetric.

We first treat the case that H is of Type (1,1). Let $\mathcal{S}$ be the restriction of $\mathcal{R}$ to H. By Theorem 9.3 we have that $\text{rot}_{\mathcal{S}}(\pi(u,v)) \in \{-\text{maxrot}(H), \ldots, \text{maxrot}(H)\}$ and hence, again by Theorem 9.3, there exists a valid orthogonal representation $\mathcal{S}'$ of H' with $\text{rot}(\pi(u',v')) = \text{rot}(\pi(u,v))$. Since H is of Type (1,1) we have that $\text{rot}_{\mathcal{S}'}(u') = \text{rot}_{\mathcal{S}}(u)$, $\text{rot}_{\mathcal{S}'}(v') = \text{rot}_{\mathcal{S}}(v)$, $\text{rot}_{\mathcal{S}'}(\pi(u',v')) = \text{rot}_{\mathcal{S}}(\pi(u,v))$ and $\text{rot}_{\mathcal{S}'}(\pi(v',u')) = \text{rot}_{\mathcal{S}}(\pi(v,u))$. Hence by plugging $\mathcal{S}'$ into the restriction of the orthogonal embedding $\mathcal{R}$ to G^- we obtain the desired embedding $\mathcal{R}'$ of G'.

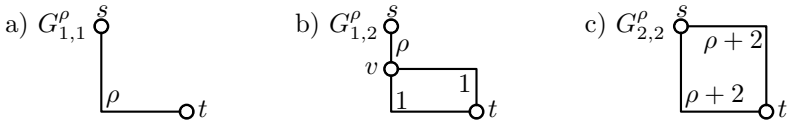


Figure 9.6.: Gadgets for st-graphs with maximum rotation ρ depending on the Type.

In the case where H is of Type (1,2) we can assume that u has degree 2 and $\deg(v) = 1$. Then the angle at u in f_i is 90° or 180° where f_i is the inner face of H incident to u. If this angle is 90°, that is, $\mathcal{S}$ is tight, we replace it by a corresponding tight embedding of H' with the same rotation, which exists by Theorem 9.3. For the case where we have an angle of 180° at u in f_i we show how to construct an orthogonal representation $\mathcal{R}''$ of G having the same planar embedding as $\mathcal{R}$ such that $\text{rot}_{\mathcal{R}''}(\pi(s,t)) = \text{rot}_{\mathcal{R}}(\pi(s,t))$ and the angle at u in f_i is 90°. Then $\mathcal{R}'$ can be constructed from $\mathcal{R}''$ as above.

By Theorem 9.3 there exists a valid and tight orthogonal representation $\mathcal{S}''$ of H with either $\text{rot}_{\mathcal{S}''}(\pi(u,v)) = \text{rot}_{\mathcal{S}}(\pi(u,v))$ or $\text{rot}_{\mathcal{S}''}(\pi(v,u)) = \text{rot}_{\mathcal{S}}(\pi(v,u))$. Without loss of generality assume the former, the other case is symmetric. Since we have increased the outer angle at u we have that $\text{rot}_{\mathcal{S}''}(u) = \text{rot}_{\mathcal{S}}(u) - 1$ and hence $\text{rot}_{\mathcal{S}''}(\pi(v,u)) = \text{rot}_{\mathcal{S}}(\pi(v,u)) + 1$. Let f_ℓ and f_r be the faces in G whose boundaries contain $\pi(u,v)$ and $\pi(v,u)$, respectively. Then we obtain $\mathcal{R}''$ by plugging $\mathcal{S}''$ into the restriction of $\mathcal{R}$ to G^- such that the angle at u in f_r is increased by 90° to 180°. Since the angle at u in f_i was decreased by 90° the sum of angles around u remains 360°. Additionally, by increasing the angle at u in f_r, its rotation is decreased by 1 which compensates the increased rotation along $\pi(v,u)$. Hence $\mathcal{R}''$ is the claimed orthogonal representation. This finishes the treatment of graphs of Type (1,2). Graphs of Type (2,2) can be treated analogously. □

We now present three especially simple families of replacement graphs, called *gadgets*, for st-graphs of Types (1,1), (1,2) and (2,2), respectively; see Figure 9.6. Let ρ be an integer. The graph $G^{\rho}_{1,1}$ is simply an edge st with $\text{flex}(st) = \rho$. The graph $G^{\rho}_{1,2}$ has three vertices s, v, t and two edges between s and v, both with flexibility 1, and the edge vt with flexibility ρ. The gadget $G^{\rho}_{2,2}$ consists of two parallel edges between s and t, both with flexibility $\rho + 2$. Note that by Corollary 9.1 all edges of our gadgets have again positive flexibility and that $\text{maxrot}(G^{\rho}_{1,1}) = \text{maxrot}(G^{\rho}_{1,2}) = \text{maxrot}(G^{\rho}_{2,2}) = \rho$. Moreover, each of these graphs has a unique embedding with s and t on the outer face.

We now describe an algorithm that computes $\text{maxrot}(G)$ for a given st-graph G with positive flexibility or decides that G does not admit a valid orthogonal representation. We use the SPQR-tree $\mathcal{T}$ of $G + st$, rooted at the Q-node corresponding to st to represent all planar embeddings of G with s and t on the outer face. Our algorithm processes the nodes of the SPQR-tree in a bottom-up fashion and computes the maximum rotation of each pertinent graph from the maximum rotations of the pertinent graphs of its children. For each node μ we maintain a variable $\text{maxrot}(\mu)$. We will prove later that after processing a node we have that $\text{maxrot}(\mu) = \text{maxrot}(\text{pert}(\mu))$. For each Q-node μ we initialize $\text{maxrot}(\mu)$ to be the flexibility of the corresponding edge. We now show how to compute $\text{maxrot}(\mu)$ from the maximum rotations of its children. We make a case distinction based on the type of μ.

If μ **is an R-node** let $\mu_1, \ldots, \mu_k$ be the children of μ. Each virtual edge in $\text{skel}(\mu)$ represents at least one incidence of an edge of G to its poles. Since $\text{skel}(\mu)$ is 3-connected each node has at least degree 3 and hence no virtual edge can represent more than two incidences, that is, the nodes of $\text{skel}(\mu)$ have degree at most 2 in the subgraphs of G that

are represented by the virtual edges of μ. As we already know their maximum rotations we can simply replace each of the graphs by a corresponding gadget; we call the resulting graph G_μ. Since the embeddings of all gadgets are completely symmetric it is enough to compute the maximum rotations of G_μ for the only two embeddings $\mathcal{E}_1$ and $\mathcal{E}_2$ induced by the embeddings of skel(μ). We set

$$\text{maxrot}(\mu) = \max\{\text{maxrot}_{\mathcal{E}_1}(G_\mu), \text{maxrot}_{\mathcal{E}_2}(G_\mu)\}$$

if one of them admits a valid representation. Otherwise we stop and return "infeasible".

If μ **is a P-node** we treat μ similar as in the case where μ is an R-node. Again, each pole has degree at least 3 in skel(μ) and hence no virtual edge can represent more than two edge incidences. We replace each virtual edge with the corresponding gadget and try all possible embeddings of skel(μ), which are at most six, and store the maximum rotation or stop if none of the embeddings admits a valid representation.

If μ **is an S-node** let $\mu_1, \ldots, \mu_k$ be the children of μ. We set

$$\text{maxrot}(\mu) = \sum_{i=1}^{k} \text{maxrot}(\mu_i) + k - 1\,.$$

Theorem 9.4. *Given an st-graph $G = (V, E)$ with positive flexibility it can be checked in $O(n^{3/2})$ time whether G admits a valid orthogonal representation. In the positive case* maxrot(G) *can be computed within the same time complexity.*

Proof. We prove that after the algorithm has processed node μ the invariant maxrot(μ) = maxrot(pert(μ)) holds. The proof is by induction on the height h of the SPQR-tree $\mathcal{T}$ of $G + st$. Let μ be the node of $\mathcal{T}$ whose parent corresponds to st.

If $h = 1$ then G is a single edge e and μ its corresponding Q-node. Since maxrot(G) equals flex(e) the claim holds. For $h > 1$ let $\mu_1, \ldots, \mu_k$ be the children of μ. By induction we have that maxrot(μ_i) = maxrot(pert(μ_i)) for $i = 1, \ldots, k$. We make a case distinction based on the type of μ.

If μ is an R- or a P-node, then by Lemma 9.4 we have maxrot(G_μ) = maxrot(pert(μ)) and since the gadgets have a unique embedding we consider all relevant embeddings of G_μ. If none of the embeddings admits a valid orthogonal representation then obviously also pert(μ) and thus G do not admit valid orthogonal representations.

If μ is an S-node and the pertinent graphs of its children admit a valid orthogonal representation then there always exists a valid orthogonal representation of pert(μ). Let $H_1, \ldots, H_k$ be the pertinent graphs of the children of μ and let $v_1, \ldots, v_{k+1}$ be the vertices in skel(μ) such that v_i and v_{i+1} are the poles of H_i. By Theorem 9.3 there exist tight orthogonal representations $\mathcal{R}_1, \ldots, \mathcal{R}_k$ of $H_1, \ldots, H_k$ with rot($\pi(v_i, v_{i+1})$) = maxrot(μ_i). We put these orthogonal representations together such that the angles at the nodes $v_2, \ldots, v_k$ on $\pi(v_1, v_{k+1})$ are 90°. Hence we get an orthogonal representation of pert(μ) with rot($\pi(v_1, v_{k+1})$) = $\sum_{i=1}^{k}$ maxrot(μ_i)$+k-1$. On the other hand if we had an orthogonal representation of pert(μ) with a higher rotation then at least one of its children μ_i would need to have a rotation that is bigger than maxrot(μ_i).

This proves the correctness of the algorithm. For the running time note that the SPQR-tree can be computed in linear time [GM00]. Computing maxrot(μ) for a given node μ from the maximum rotations of its children takes $O(|\text{skel}(\mu)|^{3/2})$ time by Theorem 9.4 since skel(μ) has only a constant number of embeddings. The total running-time follows from the fact that the total size of all skeletons is linear. □

This theorem can be used to solve FlexDraw for biconnected 4-planar graphs with positive flexibility. Such a graph G admits a valid orthogonal representation if and only if one of the graphs $G-e$, $e \in E(G)$ (which is an st-graph with respect to the endpoints of e) admits a valid orthogonal representation such that e can be added to this representation. We claim that this is possible if and only if $\text{maxrot}(G-e) + \text{flex}(e) \geq 2$. This can be seen as follows. Let s and t be the endpoints of e. Adding e to $G-e$ creates a new interior face and the total rotation of this new face needs to be 4. We can have at most two 90° angles at s and t, and we hence need to embed $G-e$ and e such that $\text{rot}(\pi(s,t)) + \text{rot}(e) = 2$. This implies that $\text{maxrot}(G-e) + \text{flex}(e) \geq 2$ is a necessary condition.

On the other hand, it is not hard to see that it is possible to add e to a tight orthogonal representation of $G-e$. First of all, if $G-e$ has a valid orthogonal representation with s and t on the outer face, then it also has a tight representation with this rotation. This implies $rot(s) \leq 0$ and $rot(t) \leq 0$. We therefore have $rot(\pi(s,t)) + \text{rot}(\pi(t,s) \geq rot(\pi(s,t)) + \text{rot}(\pi(t,s) + rot(s) + rot(t) = -4$, where the equality stems from the fact that $\pi(s,t)$ and $\pi(t,s)$ together bound the outer face. Hence we have $\text{maxrot}(G-e) \geq \max\{\text{rot}(\pi(s,t)), \text{rot}(\pi(t,s)\} \geq -2$. If $\text{flex}(e) \geq 4$, the edge e can thus be added to a tight representation with $\text{rot}(\pi(s,t)) = -2$.

If $\text{flex}(e) \leq 3$, we get $\text{maxrot}(G) \geq 2 - \text{flex}(e) \geq -1$, and thus using Theorem 9.1 we can find a tight orthogonal representation with $\text{rot}(\pi(s,t)) = 2 - \text{flex}(e)$, to which e can be added so that the new internal face bounded by e and $\pi(s,t)$ has a total rotation of 4, and thus forms a valid orthogonal representation of G. We obtain the following theorem; the running time is due to $O(n)$ applications of the algorithm for st-graphs.

Theorem 9.5. FlexDraw *can be solved in time $O(n^{5/2})$ for biconnected 4-planar graphs with positive flexibility.*

9.5. Quadratic-Time Implementation

In this Section we improve the running time of the algorithm for the biconnected case to $O(n^2)$. In the previous section we have shown that checking whether a biconnected 4-planar graph G admits a valid drawing with a given edge e on the external face can be done in time $O(n^{3/2})$. Recall that the running time stems from the fact that for each embedding $\mathcal{E}$ of each skeleton μ of the SPQR-tree we have to compute the maximum rotation of its pertinent graph with respect to this embedding. For a fixed embedding this is done by a two-step process, as in Theorem 9.2. We first compute in $O(|\text{skel}(\mu)|^{3/2})$ time an arbitrary feasible flow in an instance of Tamassia's flow network for the skeleton where some of the edges are replaced by gadgets. We call this a *base flow*. In a second step, we then compute a maximum flow in the residual network with respect to the base flow in $O(|\text{skel}(\mu)|)$ time. The running time is hence dominated by the computation of base flows.

To obtain an algorithm for the biconnected case, we simply try every edge as reference edge that has to lie on the external face, resulting in $O(n)$ applications of the above algorithm. However, when we choose a new root of the SPQR-tree and perform the traversal of the SPQR-tree, a lot of information that was already acquired in previous iterations is recomputed. In this section we show that information computed in different traversals of the SPQR-tree can be reused to improve the time that is required to compute base flows to $O(n^2)$ total time, which improves the running time of the algorithm for the biconnected case to $O(n^2)$.

Let G be a biconnected 4-planar graph with a positive flexibility function flex and let μ be a node of the SPQR-tree of G with embedding $\mathcal{E}$. Note that in this section we consider two embeddings to be equal if they differ only by the choice of the external face. Let further e be the reference edge of skel(μ), that is, the edge that μ shares with its parent. Then let $G(\mu, \mathcal{E}, e)$ be the skeleton graph of G with embedding $\mathcal{E}$ where all edges except for e are replaced by the corresponding gadget according to the maximum rotation of their expansion graphs, as used by the algorithm of the previous section. Our goal is to reuse computed flow information that was computed for $G(\mu, \mathcal{E}, e)$ when processing $G(\mu, \mathcal{E}, e')$ where e' is a different edge of skel(μ) serving as the reference edge. To this end, we define a set of operations on such graphs that allow us to transform $G(\mu, \mathcal{E}, e)$ into $G(\mu, \mathcal{E}, e')$ in time linear in the size of skel(μ). We show that while performing these operations a flow ϕ in the flow network of $G(\mu, \mathcal{E}, e)$ can be updated to a flow ϕ' in the flow network of $G(\mu, \mathcal{E}, e')$ such that the defects of ϕ and ϕ' differ only by a constant. We will then use ϕ' as a starting point to quickly check feasibility of the flow network of $G(\mu, \mathcal{E}, e')$.

We first show that given a flow network, the knowledge of a flow with small defect in the network allows to quickly check whether a feasible flow exists and that a flow with minimum defect in an instance of Tamassia's flow network can be computed in time that is quadratic in the size of the network.

Lemma 9.5. *Given a flow network $N = (V, A, \ell, u, q)$ together with a flow ϕ, a flow ϕ' of N with minimum defect can be computed in $O(|N| \operatorname{def}(\phi))$ time.*

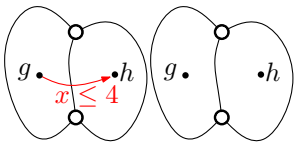
Proof. We simply apply the algorithm of Ford and Fulkerson [FF56]. We iteratively augment the flow with augmenting path from a vertex with positive defect to a vertex with negative defect. A single path of this type can be computed in $O(|N|)$ time. The algorithm stops when no such path exists. Since each such path decreases the defect of the current flow by 2 the algorithm takes at most $\operatorname{def}(\phi)$ iterations. □

Corollary 9.2. *Let $G = (V, E)$ be a 4-planar graph with n vertices and a fixed embedding. A flow with minimum defect in the corresponding flow network of Tamassia can be computed in $O(n^2)$ time.*

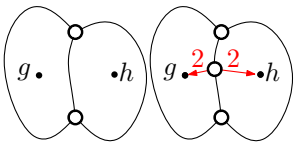
Proof. For each vertex v denote the set of incident faces by F_v. We define an initial flow ϕ by setting $\phi(v, f) = 0$ for each v and each $f \in F_v$. All other arcs receive a flow of 0. As the total amount of demands is in $O(n)$ for Tamassia's flow network the claim follows from Lemma 9.5 □

Next we define the operations that allow us to transform instances of Tamassia's flow network stemming from the skeleton of the same node with the same embedding into each other while maintaining a flow with small defect. Let G be a planar graph, let $e = uv$ be an edge of G with incident faces g and h and let ϕ be a flow in Tamassia's flow network of G. We introduce the following basic operations on G, respectively on ϕ. For an illustration see Figure 9.7.

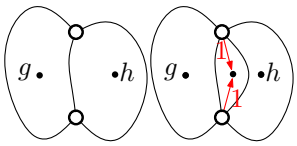
1. Setting the flow along an edge e with flow at most 4 to 0.
2. Subdividing an edge uv with flow 0 into uw and vw.
3. Doubling an edge with flow 0.
4. Removing a subdivision vertex and inserting an edge with capacity ∞ between its to neighbors.



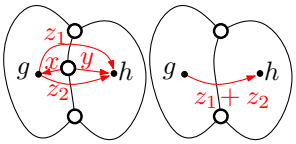
(a) Reducing flow along an edge with $\phi_e(g,h) \leq 4$ creates a defect of at most 8.



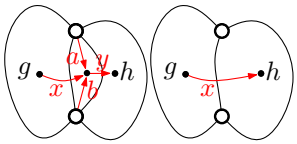
(b) Subdividing an edge e with $\phi_e(g,h) = 0$ does not increase the defect.



(c) Doubling an edge without flow increases the defect by at most 4.



(d) Removing a subdivision vertex and inserting an edge with capacity ∞ between its neighbors increases the defect by at most 12.



(e) Replacing a double edge by a single edge with capacity ∞ increases the defect by at most 12.

Figure 9.7.: Manipulation of a graph, the corresponding flow network of Tamassia and its flow. Edges of the flow network whose flow changes are shown in red. The labels indicate the amount of flow on these edges.

5. Removing one edge of a pair of double edges and setting the capacity of the remaining edge to ∞.

Lemma 9.6. *Let G be a planar graph with fixed embedding $\mathcal{E}$ and let ϕ be a flow in Tamassia's flow network of G with respect to $\mathcal{E}$.*

Let G' be a graph resulting from G by applying one of the above operations. Then a flow ϕ' in Tamassia's flow net work of G' with $\operatorname{def}(\phi') \leq \operatorname{def}(\phi) + 12$ can be computed in linear time.

Proof. The operations are illustrated in Figure 9.7. We consider the operations one by one. For operation 1 assume that e is incident to faces g and h and that $|\phi_e(g,h)| \leq 4$. To obtain ϕ' from ϕ we simply set $\phi'_e(g,h) = 0$, which clearly satisfies all lower and upper bounds. We have $\operatorname{def}_{\phi'}(g) \leq \operatorname{def}(g,f) + 4$ and $\operatorname{def}_{\phi'}(h) \leq \operatorname{def}_{\phi'}(h) + 4$. As the operation does not change the defect of any other node we have $\operatorname{def}(\phi) \leq \operatorname{def}(\phi') + 8$; see Figure 9.7a for an illustration.

For operation 2 we replace an edge $e = uv$ with incident faces g and h by two edges uw and wv. We set $f^\star$ to be a copy of f. Note that inserting the vertex increases the demands of g and h by 2. On the other hand, the new vertex w has 4 flow units. We simply route two units of flow from w to h and two units from w to g. Hence, the resulting flow $f^\star$ has the same defect as f; see Figure 9.7b.

For operation 3, see Figure 9.7c, we double the edge $e = uv$. Let k be the new face that is bounded by the two parallel edges. Assume that e_1 separates g and k and e_2 separates k and h. We route one unit of flow from each of u and v to k. This increases the defects of u and v by at most 1, each. The resulting function is a flow, as all capacity restrictions hold. Moreover, since the demand of k is 0, its defect is 2. All other nodes keep their defects and we have $\operatorname{def}(f^\star) \leq \operatorname{def}(f) + 4$.

We now show how to remove a subdivision vertex w and connect its neighbors u and v by an edge with capacity ∞; see Figure 9.7d. Let $z_1 = \phi_{uw}(g,h)$ and let $z_2 = \phi_{wv}(g,h)$ be the amounts of flow from g to h via uw and wv, respectively. Note that these values

are negative if there is flow from h to g. Without loss of generality we can assume that $z_1 + z_2 \geq 0$ holds, otherwise we simply swap g and h. Moreover, we can assume that both z_1 and z_2 are non-negative; otherwise we could reduce one of them to zero by eliminating a cyclic flow.

To obtain ϕ' from ϕ we remove all flow along the arcs wg and wh. This increases the defect of g and h by at most 4, each since the arcs wg and wh can have at most a flow of 4. Then, we replace the path uvw by the single edge e and set $\phi'(g, e, h) = z_1 + z_2$. This does not change the amount of flow any vertex sends or receives and it reduces the demands of g and h by 2, each. Hence the defect of the resulting flow ϕ' is at most $\mathrm{def}(\phi) + 12$.

Finally, we show how to remove a double edge, that is, how to implement operation 5; see Figure 9.7e. Let k be the face bounded by the double edge and let g be the other face incident to e_1 and, analogously, h the other face incident to e_2. To obtain ϕ' we first reduce the flows along the arcs vk and wk to 0, which increases the defects of v and w by at most 4, each. Next, we increase the capacity of e_1 and e_2 to ∞ and move flow along the edge from k to h (or vice versa) such that the defect of k is 0. Essentially, we shift the defect of k to h. Now, since the defect and the demand of k are both 0, we have that $\phi_{e_1}(g, k) = \phi_{e_2}(k, h)$. Consequently, removing the edge e_2 and setting $\phi'_{e_1}(g, h) = \phi_{e_1}(g, k)$ does not change the defects of g and h. Thus, the total increase of defect is at most 8. □

Note that a sequence of these operations can be used to replace a gadget by a single edge and vice versa. The following lemma states that this is always possible while increasing the defect of a given flow only by a certain constant.

Lemma 9.7. *Given $G(\mu, \mathcal{E}, e)$ together with a flow ϕ in its corresponding flow network in which e has at most 4 units of flow, a flow ϕ' in the flow network of $G(\mu, \mathcal{E}, e')$ with $\mathrm{def}(\phi') \leq \mathrm{def}(\phi) + 88$ can be computed in linear time.*

Proof. To transform $G(\mu, \mathcal{E}, e)$ into $G(\mu, \mathcal{E}, e')$ we first replace e by the corresponding gadget representing its expansion graph. Then we transform the gadget that represents e' in $G(\mu, \mathcal{E}, e)$ into the single edge e'. Finally, we may have to change the outer face. We now show that all these operations can be performed while maintaining the claimed flow ϕ'. We start with $\phi = \phi'$.

The cost of the first step depends on the Type of the expansion graph of e. If e is of Type (1,1), no change is necessary. For Type (1,2) we reduce the flow along e, subdivide e and double one of the resulting edges, for Type(2,2) we first reduce the flow of e and then double it. In all cases we apply Lemma 9.6 at most three times and the increase of defect is at most 36. Similarly, we obtain a defect of at most 36 for the second step, thus yielding a flow ϕ' with $\mathrm{def}(\phi') \leq \mathrm{def}(\phi) + 72$. Finally, by the definition of Tamassia's flow network, to change the outer face to an interior face we increase its demand by 8 and we decrease the demand of an inner face by 8 to make it the outer face. The defect of ϕ' in the resulting flow network is therefore at most 88. □

We apply this lemma to save computing time when processing the graph $G(\mu, \mathcal{E}, e)$. If we process μ for the first time with embedding $\mathcal{E}$ (embeddings are considered equal if they only differ by the choice of the external face), using Lemma 9.2, we compute in $O(|\mathrm{skel}(\mu)|^2)$ time a flow ϕ with minimum defect in the corresponding flow network, where the flexibility of e it set to 4. We now distinguish several cases.

If ϕ has defect 0, it is a valid solution and we have found our base flow. Moreover, we store ϕ along with μ and $\mathcal{E}$ for future reuse. If the defect of ϕ is greater than 0 and at most 88, we know that μ does not admit a valid drawing with embedding $\mathcal{E}$. Again, we

store ϕ for future reuse. If the defect of ϕ is greater than 88, we store the information that μ does not admit a valid drawing with embedding $\mathcal{E}$, independent of the choice of the reference edge and the outer face. This is true since otherwise we could apply Lemma 9.6 to obtain a flow ϕ' for $G(\mu, \mathcal{E}, e)$ with defect at most 88, contradicting the optimality of ϕ.

Whenever we encounter μ with embedding $\mathcal{E}$ again, but this time with reference edge e', we can either conclude that it does not admit a valid drawing (if the embedding is marked as invalid for μ), or check in $O(|\text{skel}(\mu)|)$ time whether it admits a feasible flow as follows. Let ϕ be the stored flow. By applying Lemma 9.6 we can construct in time $O(|\text{skel}(\mu)|)$ a flow ϕ' in the flow network of $G(\mu, \mathcal{E}, e')$ with $\text{def}(\phi') \leq \text{def}(\phi) + 88 \leq 196$. We then apply Lemma 9.5 to compute a feasible base flow in $O(|\text{skel}(\mu)|)$ time, if it exists.

Since each node μ has only a constant number of embeddings, the total time for all base flow computations is bounded by $O(n^2)$. After that checking a node μ with a given embedding can be done in $O(|\text{skel}(\mu)|)$ time. Since each node is checked at most linearly often with each embedding, the total running time is in $O(n^2)$. We have proved the following theorem.

Theorem 9.6. FlexDraw *can be solved in $O(n^2)$ time for biconnected 4-planar graphs with positive flexibility.*

9.6. Connected Graphs

In this section we generalize our results to connected 4-planar graphs that are not necessarily biconnected. We analyze the conditions under which orthogonal representations sharing a cut vertex can be combined and use the block-cutvertex tree to derive an algorithm that decides whether a connected 4-planar graph with positive flexibility admits a valid orthogonal drawing.

Lemma 9.8. *Let G be a connected 4-planar graph with cutvertex v and corresponding cut components $H_1, \ldots, H_k$. Then G admits a valid orthogonal representation if and only if all cut components H_i have valid orthogonal representations such that at most one of them has v not on the outer face.*

Proof. The only if part is clear since a valid orthogonal representation of G induces valid orthogonal representations of all cut components H_i such that at most one of them does not have v on its outer face.

Now let $\mathcal{S}_i$ be valid orthogonal representations of the cut components H_i for $i = 1, \ldots, k$ such that at most one of them does not have v on its outer face.

If all of them have v on their outer face then by Lemma 9.2 we can assume that these representations are tight. Then it is clear that the components $H_1, \ldots, H_k$ can be merged together in v maintaining their representations $\mathcal{S}_i$.

Otherwise, one of the representations, without loss of generality $\mathcal{R}_1$, does not have v on the outer face. If v has degree at least 2 in at most one of the graphs, we can simply merge the corresponding tight representations as bridges can always be added.

The only problem that can arise is that there are exactly two components H_1 and H_2, v has degree 2 in both of them, and the angles incident to v in H_1 are 180°. We resolve this situation by either increasing or decreasing the number of bends of an incident edge and changing the angles at v appropriately. □

Now let G be a connected 4-planar graph with positive flexibility and $\mathcal{B}$ its block-cutvertex tree. Let further B be a block of G that is a leaf in $\mathcal{B}$ and let v be the unique cutvertex of B.

If B is the whole graph G we return "true" if and only if G admits any valid orthogonal representation. This can be checked with the algorithm from the previous Section.

If B is not the whole graph G we check whether B admits a valid orthogonal representation having v on its outer face. This can be done with the algorithm from the previous section by rooting the SPQR-tree of B at all edges incident to v. If it does admit such an embedding then by Lemma 9.8 G admits a valid orthogonal embedding if and only if the graph G', which is obtained from G by removing the block B, admits a valid orthogonal embedding. We check G' recursively. If B does not admit such an embedding we mark B and proceed with another unmarked leaf. If we ever encounter another block B' that has to be marked we return "infeasible". This is correct as in this case B has to be embedded in the interior of B' and vice versa, which is obviously impossible. Checking a single block B requires $O(|B|^2)$ time by Theorem 9.6. Since the total size of all blocks is linear the total running-time is $O(n^2)$. This proves the following theorem, which is the main result of this paper.

Theorem 9.7. FlexDraw *can be solved in $O(n^2)$ time for 4-planar graphs with positive flexibility.*

9.7. Complexity

In this section, we consider the complexity of FlexDraw for cases that lie between 0-embeddability and the case of positive flexibility. For example, it is an interesting question, whether FlexDraw can still be solved efficiently, if the subgraph consisting only of the edges with flexibility 0 has a special structure. Since 0-embeddability can be solved in polynomial time for series-parallel graphs and maxdeg-3 graphs, one might hope for an algorithm that solves FlexDraw efficiently if the edges with flexibility 0 form a graph in one of these classes. Unfortunately, we can show that both these cases, and in fact a lot of simpler cases remain NP-hard.

The gadgets used in the NP-hardness proof of 0-embeddability by Garg and Tamassia [GT01] can be constructed in a slightly modified form such that the edges with flexibility 0 even form a matching. Thus, FlexDraw is NP-hard, even if the edges with flexibility 0 form a matching. However, the gadgets constructed in this way are slightly more flexible than the original ones, and hence all steps of the proof need to be reworked, in order to complete the NP-hardness proof for this case. We will therefore only present the gadgets, but not the full proof. Instead we will show a weaker statement, for which we have a short proof. Both constructions rely on a basic building block, which is described next.

Consider the *wheel* on five vertices, which consists of a cycle on vertices $v_2, \ldots, v_5$ and the center vertex v_1 that is connected to all other vertices; see Figure 9.8a. In the following we assume that the flexibility of each edge of the wheel is 1. A corresponding flex-drawing is shown in Figure 9.8b. We claim that the outer face, as well as its orthogonal description are unique. To see this, consider Tamassia's flow network for the wheel in the embedding shown in Figure 9.8a. Since the outer face is incident to four vertices it has a demand of 12. On the other hand, it can receive at most 8 units of flow from $v_2, \ldots, v_5$ and at

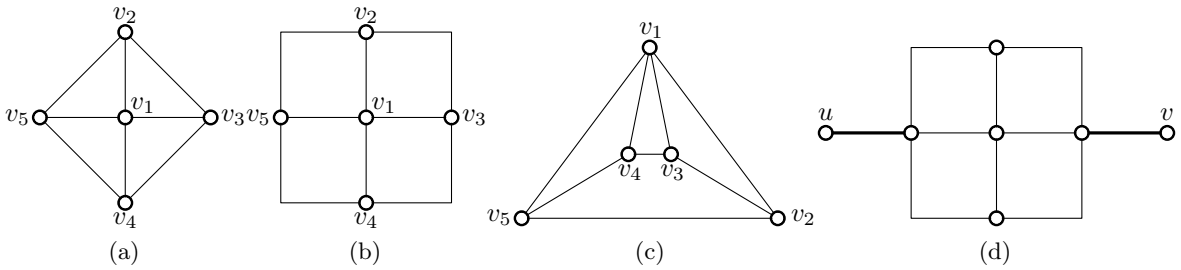


Figure 9.8.: A graph that is almost rigid graph, even if every edge has flexibility 1.

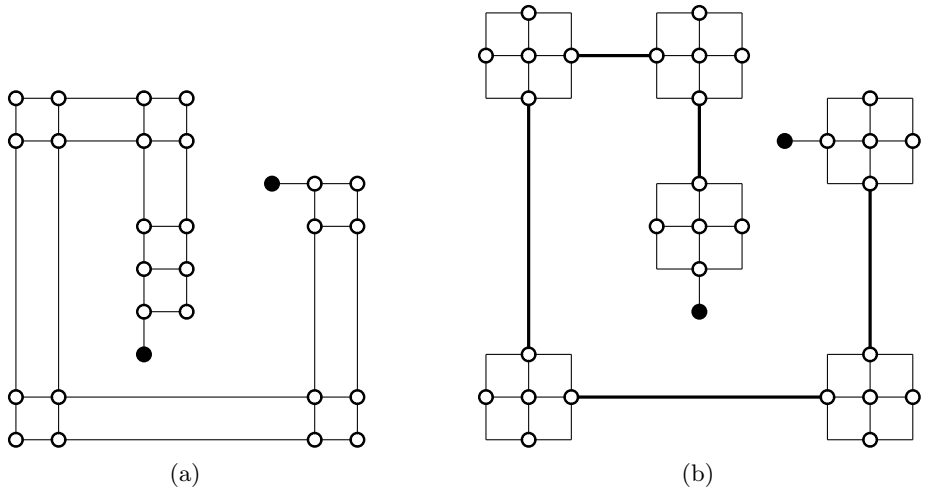


Figure 9.9.: The tendril gadget used in the proof by Garg and Tamassia [GT01], all edges have flexibility 0 (a). A substitution, where the thin edges have flexibility 1 and the thick edges, which form a matching, have flexibility 0 (b). The properties of the wheel graphs used in the second gadget guarantee that both gadgets behave very similarly.

most 4 units of flow via the incident edges. Hence, in any feasible flow the outer face must receive two units of flow from its incident vertices, and one unit of flow form each incident edge. This completely describes the outer face. The only degree of freedom is that the center vertex can be rotated by 90 degrees to the left or to the right. Moreover, all other embeddings of the wheel do not allow for a 1-bend embedding. Since the wheel is 3-connected the only embedding choice is the outer face. Up to renaming the there the only embedding that is different from the embedding in Figure 9.8a is shown in Figure 9.8c. In Tamassia's flow network the outer face has a demand of 10. However, it can receive at most one unit of flow from v_1, at most two units of flow from v_2 and v_5, and at most three units of flow via its incident edges, which adds up to a total of 8. Hence, the wheel does not admit a 1-bend drawing with this embedding.

Figure 9.9a shows the gadget that Garg and Tamassia use for their NP-hardness proof of 0-embeddability. Figure 9.9b shows the replacement we propose. The thick edges have flexibility 0, all other edges have flexibility 1. Using the knowledge about the drawings of the wheel-subgraphs, it is not hard to see that these two gadgets behave very similar. The only difference is that the edge incident to the black attachment vertices admit only two different drawings in the gadget of Garg and Tamassia, while our gadget admit three different drawings. By choosing the numbers in the reduction by Garg and Tamassia in the

right way, this small difference does not affect the overall reduction. To avoid a lengthy argument, we rather present a proof of the following weaker statement.

Theorem 9.8. FlexDraw *is NP-hard, even if the subgraph with flexibility 0 is a spanning tree or a spanning union of disjoint stars.*

Proof. We reduce from 0-embeddability, which is known to be NP-hard [GT01]. Let $G = (V, E)$ be an instance of 0-embeddability and let T be any spanning tree of G. Let G' be the graph that is obtained from G by replacing each edge $uv \in E \setminus T$ by the gadget shown in Figure 9.8d, where the two bold edges have flexibility 0 and all other edges in the gadget have flexibility 1. As in each flex-drawing the rotation between the two vertices of degree 1 is 0, it follows that G' admits a flex-drawing if and only if G admits a 0-embedding. Note that in the gadget the endpoints u and v are not connected by a path of edges with flexibility 0. Therefore, the edges with flexibility 0 in G' form a tree.

The same argument works if we replace all edges of G by gadgets to obtain G'. In this case the edges with flexibility 0 form a union of disjoint stars, each having a vertex of the original graph as its center. □

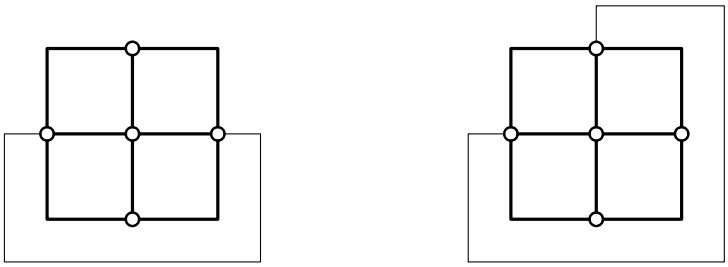
9.8. Concluding Remarks

The main result of this chapter is that FlexDraw can be solved efficiently for graphs with positive flexibility. To prove this, we first showed that the set of possible drawings of a graph with positive flexibility can be described by a single number, its maximum rotation. The fact that subgraphs can be substituted by graphs with the same maximum rotation without affecting the overall maximum rotation enabled us to gradually reduce the size of the graph that needed to be considered. This directly led to a polynomial-time algorithm, which together with a specialized out-of-the-box flow algorithm for planar graphs resulted in a running time of $O(n^{5/2})$. We then introduced flows with defects to share more information between different phases of the algorithm. Using these concepts and a careful implementation we were able to reduce the running time to $O(n^2)$.

The main result of this chapter closes the long-standing complexity gap between 0-embeddability and 2-embeddability. However, the result is much more general, as it enables us to specify the maximum number of bends for each edge, individually. This may have interesting applications in domains such as the layout of UML diagrams, which are typically drawn with orthogonal edges, and where certainly some of the edges are much more important than others, and thus should have few bends, possibly at the cost of more bends at unimportant edges. To obtain a nice drawing, it may even be desirable to specify no upper bound at all on the number of bends for unimportant edges. It is straightforward to generalize the results presented in this chapter to positive flexibility functions $\text{flex} : E \longrightarrow \mathbb{N} \cup \{\infty\}$, where some edges may be bent arbitrarily often.

We further explored the complexity gap between 0-embeddability and FlexDraw with positive flexibility. We have shown that FlexDraw is NP-hard, even if the subgraph consisting of edges with flexibility 0 forms a tree or a union of disjoint stars. We further indicated that this still holds, even if the edges with flexibility 0 form a matching.

Open questions. We leave open two questions, which we believe to be most interesting. As we have seen, FlexDraw remains NP-hard, even if the edges with flexibility 0 form a tree, or a matching. It is an interesting question whether FlexDraw can still be handled



(a) (b)

Figure 9.10.: Examples of graphs that require an edge with several bends in a flex-drawing. The wheel with an additional edge shown in (a) requires an edge with four bends in any drawing, if the bold edges have a flexibility of 1. If the embedding is fixed, five bends are necessary for the thin edge in (b).

if few edges are required to have no bends, for example, a constant number of edges or a set of independent edges.

Another interesting question stems from the problem of computing a *nice* flex-drawing. In general, 4-planar graphs admit drawings with at most 2 bends per edge, with the only exception of the octahedron, which requires an edge with three bends [BK94]. Thus, in the absence of flexibility constraints a linear number of bends is sufficient. Figure 9.10 shows that flex-drawings of graphs with positive flexibility may require edges with four bends. How many bends may be required for a flex-drawing of a graph with positive flexibility? Is there a constant C such that for each graph G with positive flexibility that admits a flex-drawing there is one with at most C bends per edge? This also has implications on the time for actually computing a flex-drawing since computing a drawing from an orthogonal representation requires time that is quadratic in the number of bends. If it was possible that a non-constant number of edges would need to have a non-constant number of bends, this would result in a total running time that is super-quadratic, and thus slower than the algorithm for finding the embedding. We suspect that this is not the case; we conjecture that generally $C = 4$, and that $C = 5$ if the embedding is fixed.

Chapter 10

Conclusion

Summary. In this thesis, we have studied a variety of graph problems related to planarity. As indicated in Chapter 1, planarity is a fundamental concept in graph theory, and has strongly influenced its development. Moreover, many applications use visualizations of graphs to present them in an understandable way. Crossings in drawings hurt readability, and thus algorithms that produce drawings with few crossings, and in particular planar drawings, are important. We have pointed out that planarity can assume different roles in a problem, and that the complexity of a problem seems to be connected to this role.

In the first part of this thesis, our main focus was on combinatorial optimization problems where planarity either occurred as an additional side constraint to an optimization problem or as an input restriction. It turned out that planarity is not a very useful property as a side constraint. It does not give too much structure to the set of feasible solutions, and many problems become harder in the presence of such constraints. Only very strong planarity conditions like convex geometric planar graphs or relaxed problem versions like path augmentation allowed us to establish more structure and efficient solution algorithms in Chapter 3. On the other hand, planarity as an input restriction, as in the matching problem in Chapter 5, proved to be tremendously helpful. It is worth noting that our algorithm for finding large matchings quickly neither just uses basic planarity properties, such as the existence of a vertex of low degree, nor does it rely on the existence of special data structures that are particularly efficient for planar graphs. Instead, it exploits planarity at a deeper level, by using a combinatorial embedding of the input graph as a guide to find edges that can be used to improve the matching.

The second part of this thesis is motivated by problems stemming from the areas of network visualization and graph drawing. A planar graph usually has many different planar drawings, and some embeddings may allow for nicer drawings than others. We have studied the complexity of finding an optimal embedding for a variety of different drawing styles. It turned out that the set of planar embeddings of a graph has a rich structure, and thus we were able to show that in many cases an optimal embedding can be computed efficiently. The main observation here is that the difficulty of these problems is strongly connected to the existence of simple and useful combinatorial characterizations of valid solutions. It is worth noting that, although all the problems we considered in the second part come from the field of graph drawing, we rarely had to work with actual drawings of graphs. Instead, almost all of the time, it was sufficient to work on higher levels of abstraction.

For testing the planarity of partially embedded (or partially drawn) graphs, the whole problem could be treated at the level of embeddings, and it turned out that the valid solutions have a very particular structure that indeed allows to resolve constraints locally at each skeleton of the SPQR-tree of the input graph. This resulted in a very simple

polynomial-time algorithm, for which correctness is almost immediate. Similarly, in Chapter 9, where we considered the problem FlexDraw, we never worked with actual orthogonal drawings. Instead, right from the start, we only worked with the already known orthogonal representations as an abstraction. On top of this, we showed that for graphs with positive flexibility, with two given vertices on the outer face, all relevant embeddings can in fact be encoded in a single number, the maximum rotation. This allowed us to address the problem FlexDraw on a very high abstraction level, simply by computing the maximum rotation of the whole graph. Afterwards, a whole machinery of combinatorial results was available to produce a corresponding solution.

Finally, in Chapter 8, we got a reminder of how friendly planarity actually is. There, we had a small glimpse of what happens when we leave the realm of planarity and consider even only slightly non-planar drawings, namely simultaneous embeddings with fixed edges of two graphs. In this case, it seems that the fact that we could use the SPQR-tree of the intersection graph to handle its embedding (and thus a large part of the embedding of each graph), helped a lot to keep control of the many different possibilities to embed the two input graphs.

Outlook. As indicated in each of the chapters, many of the results presented in this thesis lead to related questions that are theoretically interesting and are well worth studying. Here we take a broader look and raise some challenges for future research in the area of planar graphs, and beyond.

In terms of combinatorial optimization, by now there exist a lot of very efficient algorithms that are specialized to planar graphs. Still, some of the most basic problems are not yet solved in a satisfying manner. The obvious two questions in this field are to either resolve more questions on planar graphs or to extend results for planar graphs to more general graphs, in particular to *bounded genus* graphs, *k-degenerate* graphs, and graphs that admit small, balanced separators. Further, many of the very efficient algorithms for planarity are rather complicated, and it is an ongoing endeavor to simplify these algorithms and unify their descriptions while maintaining their theoretical guarantees. The recent progress in designing simple and efficient algorithms for planarity testing [HT08, Bra09] and maximum flows in planar graphs [Eri10] shows great potential and possibly leads to a deeper understanding of planarity, which may in turn enable the generalization of such algorithms to more general graphs.

Also for planar graphs, a lot of very basic questions are still open. For example, what is the complexity of determining the treewidth of a planar graph? For general graphs, this problem is NP-complete. For planar graphs, however, neither an NP-hardness proof nor a polynomial-time algorithm is known. But also for polynomial-time solvable problems a lot of challenges remain, in particular when it comes to determining their exact complexity. Until now, there is a lack of lower bounds in the area of planar graph algorithms, and essentially only linear-time algorithms are provably optimal. Not too long ago, it was shown that maximum *st*-flows in planar graphs can be computed in $O(n \log n)$ time [BK09, Eri10], which is at least optimal to within a polylogarithmic factor. For computing maximum matchings, a corresponding result is still unknown, the fastest known maximum matching algorithm for planar graphs has running time $O(n^c)$ with $c > 1$. However, for bipartite planar graphs, at least a perfect matching can be found faster, namely in $O(n \log^3 n)$ time [MN95, FR06], which gives hope that also the general planar case might be solvable in time $O(n \operatorname{polylog} n)$. Settling this question, one way or the other, would mean a huge leap forward in determining the exact complexity of these problems.

Concerning graph drawing and network visualization, one future challenge rises from

the fact that in practice graphs are often not static, but change dynamically over time. In this case it is important to visualize them in such a way that the user's mental map is preserved – in order to facilitate the recognition of stable substructures and to allow for easy orientation. The work on partially embedded graphs in this thesis can be seen as a step in this direction. A typical example of such graphs are UML diagrams that are commonly used for describing software architectures, and thus change over time as the software evolves. These diagrams are typically drawn in an orthogonal drawing style, where the vertices are represented by large boxes. It seems worthwhile to investigate whether combining the ideas of drawing important edges with fewer bends, and that of preserving as much of a previous drawing as possible yields good algorithms for drawing UML diagrams in a dynamic setting.

Given the crucial role that planarity has played in the development of graph theory and its fundamental importance in graph drawing, it would probably be an exaggeration to claim that planarity will become more important in the future. Rather I believe that planarity will keep its importance, in particular as a stepping stone for developing algorithms for larger, more general classes of graphs.

Whereas planar graphs form a large class of graphs with a rich structure, one should not forget that many applications involve graphs that are not planar. Fortunately, many real-world graphs are sparse, and in many cases in some sense close to planar. Clearly, there is a strong demand for efficient algorithms to handle such graphs. Several notions for describing these "slightly non-planar" graphs have been suggested, for example *quasi-planar* graphs (graphs that can be drawn in the plane such that no three of its edges are pairwise crossing), *almost-planar* graphs (graphs for which for each edge e at least one of $G - e$ and G/e is planar), *k-planar* graphs (graphs that can be drawn such that each edge has at most k crossings), and recently *RAC* graphs (graphs that have straight-line drawings such that all crossings form a right angle). In particular in the area of graph drawing, as the large number of papers on RAC graphs and RAC drawings at the last two graph drawing conferences shows, a shift towards studying and understanding these non-planar graphs has already started.

I believe that it will become a major challenge of the coming years to develop a theoretical toolbox for handling such close-to-planar graphs, and to adapt, enhance and generalize as many algorithms for planar graphs as possible to these larger graph classes.

Bibliography

[ADF+10a] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. In *Proceedings 21st ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 202–221. SIAM, 2010. [see page 81]

[ADF+10b] Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Testing the simultaneous embeddability of two graphs whose intersection is a biconnected graph or a tree. In C. S. Iliopoulos and W. F. Smyth, editors, *Proc. 21st Internat. Workshop Combinatorial Algorithms (IWOCA'10)*, volume 6460 of *Lecture Notes Comput. Sci.*, pages 212–225. Springer-Verlag, 2010. [see page 165]

[AGH+08] Manuel Abellanas, Alfredo García, Ferran Hurtado, Javier Tejel, and Jorge Urrutia. Augmenting the connectivity of geometric graphs. *Comput. Geom. Theory Appl.*, 40(3):220–230, 2008. [see pages 24, 25, 39, and 42]

[AGKN10] Patrizio Angelini, Markus Geyer, Michael Kaufmann, and Daneiel Neuwirth. On a tree and a path with no geometric simultaneous embedding. In Ulrik Brandes and Sabine Cornelsen, editors, *Proc. 18th Internat. Sympos. Graph Drawing (GD '10)*, volume 6502 of *Lecture Notes Comput. Sci.*, pages 38–49. Springer-Verlag, 2010. [see page 166]

[AJIR+09] Marwan Al-Jubeh, Mashhood Ishaque, Kristóf Rédei, Diane L. Souvaine, and Csaba D. Tóth. Tri-edge-connectivity augmentation for planar straight line graphs. In Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra, editors, *Proc. 20th Internat. Sympos. Algorithms Comput. (ISAAC'09)*, volume 5878 of *Lecture Notes Comput. Sci.*, pages 902–912. Springer-Verlag, 2009. [see page 25]

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *The Annals of Mathematics, Second Series*, 160(2):781–793, 2004. [see page 17]

[Alg07] Algorithmic Solutions. The LEDA user manual version 5.2. `www.algorithmic-solutions.info/leda_manual`, visited 07/04/2007. [see page 65]

[APT79] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979. [see pages 18, 176, and 183]

[BBDL01] Therese Biedl, Prosenjit Bose, Eric Demaine, and Anna Lubiw. Efficient algorithms for Petersen's theorem. *J. Algorithms*, 38:110–134, 2001. [see page 66]

[BCD+07] Peter Braß, Eowyn Cenek, Christian A. Duncan, Alon Efrat, Cesim Erten, Dan P. Ismailescu, Stephen G. Kobourov, Anna Lubiw, and Joseph S. B. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry*, 36(2):117–130, 2007. [see page 166]

[BDD00] Paola Bertolazzi, Giuseppe Di Battista, and Walter Didimo. Computing orthogonal drawings with the minimum number of bends. *IEEE Trans. Computers*, 49(8):826–840, 2000. [see page 86]

[BDD+04] Therese Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Math.*, 285(1–3):7–15, 2004. [see page 66]

[BK94] Therese Biedl and Goos Kant. A better heuristic for orthogonal graph drawings. In *Proc. 2nd Europ. Symp. Algorithms (ESA'94)*, volume 855 of *Lecture Notes Comput. Sci.*, pages 24–35. Springer-Verlag, 1994. [see pages viii, 192, 193, and 212]

[BK09] Glencora Borradaile and Philip Klein. An $o(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2), 2009. [see pages 2 and 214]

[BKRW11] Thomas Bläsius, Marcus Krug, Ignaz Rutter, and Dorothea Wagner. Orthogonal graph drawing with flexibility constraints. In U. Brandes and S. Cornelsen, editors, *Proc. 18th Internat. Sympos. Graph Drawing (GD'10)*, volume 6502 of *Lecture Notes Comput. Sci.*, pages 92–104. Springer-Verlag, 2011. [see page 191]

[BM89] Daniel Bienstock and Clyde L. Monma. Optimal enclosing regions in planar graphs. *Networks*, 19:79–94, 1989. [see page 12]

[BM90] Daniel Bienstock and Clyde L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, 5(1):93–109, 1990. [see page 12]

[BM04] John M. Boyer and Wendy J. Myrvold. On the cutting edge: simplified $O(n)$ planarity by edge addition. *J. Graph Alg. Appl.*, 8(3):241–273, 2004. [see pages 2 and 11]

[Bra09] Ulrik Brandes. The left-right planarity test. manuscript, 2009. [see page 214]

[CGMW09] Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Christian Wolf. Inserting a vertex into a planar graph. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 375–383, Philadelphia, PA, USA, 2009. SIAM. [see page 193]

[Cha91] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(1):485–524, 1991. [see page 42]

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2nd edition, 2001. [see pages 7 and 38]

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annu. ACM Sympos. Theor. Comput. (STOC'71)*, pages 151–158. ACM, 1971. [see page 18]

[Coo03] Matthew Cook. Still Life Theory. In C. Moore and D. Griffeath, editors, *New Constructions in Cellular Automata*, volume 226, pages 93–118. Oxford University Press, 2003. [see page 47]

[Cor88] Gerard Cornuéjols. General factors of graphs. *Journal of Combinatorial Theory, Series B*, 45:185–198, 1988. [see page 59]

[COS01] Richard Cole, Kirstin Ost, and Stefan Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21, 2001. [see page 66]

[CW87] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proc. 19th Annu. ACM Conf. Theory Comput. (STOC'87)*, pages 1–6, 1987. [see page 65]

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008. [see page 40]

[dBK10] Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In *Proc. 16th International Computing and Combinatorics Conference (COCOON'2010)*, volume 6196 of *Lecture Notes Comput. Sci.*, pages 216–225. Springer-Verlag, 2010. [see page 51]

[dFdMR06] Hubert de Fraysseix, Patrice O. de Mendez, and Pierre Rosenstiehl. Trémaux trees and planarity. *Int. J. Found. Comput. Sci.*, 17:1017–1030, 2006. [see pages 2 and 11]

[Die10] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Heidelberg, 4th edition edition, 2010. [see page 7]

[DL07a] Emilio Di Giacomo and Giuseppe Liotta. Simultaneous embedding of outerplanar graphs, paths, and cycles. *Int. J. Comput. Geom. Appl.*, 17(2):139–160, 2007. [see page 114]

[DL07b] Emilio Di Giacomo and Giuseppe Liotta. Simultaneous embedding of outerplanar graphs, paths, and cycles. *Int. J. Computational Geometry and Applications*, 17(2):139–160, 2007. [see page 166]

[DLV98] Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998. [see pages 192 and 195]

[DMP64] G. Demoucron, Y. Malgrange, and R. Pertuiset. Reconnaissance et construction de représentations planaires topologiques. *Rev. Franc. Rech. Oper.*, 8:33–34, 1964. [see page 147]

[Dor02] Christoph Dornheim. Planar graphs with topological constraints. *J. Graph Alg. Appl.*, 6(1):27–66, 2002. [see page 82]

[DT96a] Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM J. Comput.*, 25(5):956–997, 1996. [see pages 12, 81, and 82]

[DT96b] Giuseppe DiBattista and Roberto Tamassia. On-line maintenance of triconnected components with spqr-trees. *Algorithmica*, 15:302–318, 1996. [see page 12]

[EBGJ+07a] Alejandro Estrella-Balderrama, Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous geometric graph embeddings. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *GD '07*, volume 4875 of *Lecture Notes Comput. Sci.*, pages 280–290, 2007. [see page 114]

[EBGJ+07b] Alejandro Estrella-Balderrama, Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous geometric graph embeddings. In S. H. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing (GD '07)*, volume 4875 of *LNCS*, pages 280–290, 2007. [see page 166]

[Edm69] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications*, pages 69–87, Calgary, 1969. [see page 53]

[EK04] Cesim Erten and Stephen G. Kobourov. Simultaneous embedding of planar graphs with few bends. In J. Pach, editor, *GD '04*, volume 3383 of *Lecture Notes Comput. Sci.*, pages 195–205, 2004. [see page 114]

[EK05] Cesim Erten and Stephen G. Kobourov. Simultaneous embedding of planar graphs with few bends. *J. Graph Algorithms Appl.*, 9(3):347–364, 2005. [see page 166]

[Eri10] Jeff Erickson. Maximum flows and parametric shortest paths in planar graphs. In *Proceedings 21st ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 794–804. SIAM, 2010. [see pages 2 and 214]

[ET76] Kapali P. Eswaran and Robert E. Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4):653–665, 1976. [see page 24]

[FF56] Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 15(4):302–318, 1956. [see pages 195 and 205]

[FGJ+08a] J. J. Fowler, C. Gutwenger, M. Jünger, P. Mutzel, and M. Schulz. An SPQR-tree approach to decide special cases of simultaneous embedding with fixed edges. In I. G. Tollis and M. Patrignani, editors, *Graph Drawing (GD '08)*, volume 5417 of *Lecture Notes Comput. Sci.*, pages 157–168, 2008. [see page 166]

[FGJ+08b] Joseph Fowler, Carsten Gutwenger, Michael Jünger, Petra Mutzel, and Michael Schulz. An SPQR-tree approach to decide special cases of simultaneous embedding with fixed edges. In I. G. Tollis and M. Patrignani, editors, *GD '08*, volume 5417 of *Lecture Notes Comput. Sci.*, pages 157–168, 2008. [see page 114]

[Fia03] Jiří Fiala. NP-completeness of the edge precoloring extension problem on bipartite graphs. *J. Graph Theory*, 43(2):156–160, 2003. [see page 82]

[FJ81] Greg N. Frederickson and Joseph Ja'Ja'. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981. [see pages 24 and 46]

[FJKS08] Joseph Fowler, Michael Jünger, Stephen G. Kobourov, and Michael Schulz. Characterizations of restricted pairs of planar graphs allowing simultaneous embedding with fixed edges. In H. Broersma, T. Erlebach, T. Friedetzky, and D. Paulusma, editors, *WG '08*, volume 5344 of *Lecture Notes Comput. Sci.*, pages 146–158, 2008. [see pages 114 and 166]

[FM98] Sergej Fialko and Petra Mutzel. A new approximation algorithm for the planar augmentation problem. In *Proc. 9th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'98)*, pages 260–269, 1998. [see page 24]

[FR06] Jittat Fakcharoenphol and Satish Rao. Planar graphs, negative weight, shortest paths, and near linear time. *J. Comput. System Sci.*, 72:868–889, 2006. [see pages 66 and 214]

[Fra06] Fabrizio Frati. Embedding graphs simultaneously with fixed edges. In M. Kaufmann and D. Wagner, editors, *GD '06*, volume 4372 of *Lecture Notes Comput. Sci.*, pages 108–113, 2006. [see pages 114 and 166]

[FRW10] Robert Franke, Ignaz Rutter, and Dorothea Wagner. Computing large matchings in planar graphs with fixed minimum degree. *Theoret. Comput. Sci.*, 2010. available online, `http://dx.doi.org/10.1016/j.tcs.2010.06.012`. [see page 65]

[GJ77] Michael R. Garey and David S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977. [see page 113]

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979. [see pages 7, 18, 19, and 50]

[GJP+06] Elisabeth Gassner, Michael Jünger, Merijam Percan, Marcus Schaefer, and Michael Schulz. Simultaneous graph embeddings with fixed edges. In F. V. Fomin, editor, *WG '06*, volume 4271 of *Lecture Notes Comput. Sci.*, pages 325–335, 2006. [see pages 114 and 166]

[GK04] Jan F. Groote and M. K. Keinänen. Solving disjunctive/conjunctive boolean equation systems with alternating fixed points. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes Comput. Sci.*, pages 436–450. Springer-Verlag, 2004. [see page 48]

[GKM08] Carsten Gutwenger, Karsten Klein, and Petra Mutzel. Planarity testing and optimal edge insertion with embedding constraints. *J. Graph Algorithms Appl.*, 12(1):73–95, 2008. [see pages 82 and 116]

[GKT01] Harold N. Gabow, Haim Kaplan, and Robert E. Tarjan. Unique maximum matching algorithms. *J. Algorithms*, 40(2):159–183, 2001. [see page 66]

[GKV09] Markus Geyer, Michael Kaufmann, and Imrich Vrt'o. Two trees which are self-intersecting when drawn simultaneously. *Discrete Math.*, 307(4):1909–1916, 2009. [see page 166]

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, chapter 9: Stable Sets in Graphs, pages 273–303. Springer-Verlag, 1988. [see page 82]

[GM00] Carsten Gutwenger and Petra Mutzel. A linear time implementation of SPQR-trees. In J. Marks, editor, *GD '99*, volume 1984 of *Lecture Notes Comput. Sci.*, pages 77–90, 2000. [see pages 16, 86, 87, 96, 168, and 203]

[GMW01] Carsten Gutwenger, Petra Mutzel, and René Weiskircher. Inserting an edge into a planar graph. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 246–255, Philadelphia, PA, USA, 2001. SIAM. [see page 193]

[GP08] Jan F. Groote and Bas Ploeger. Switching graphs. In *Proceedings of the 2nd Workshop on Reachability Problems (RP'2008)*, ENTCS, pages 119–135, 2008. [see pages 47 and 48]

[GT85] Harold N. Gabow and Robert E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30:209–221, 1985. [see page 179]

[GT01] Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. [see pages viii, 192, 193, 209, 210, and 211]

[Hal35] Philip Hall. On representatives of subsets. *Jour. London Math. Soc.*, 10:26–30, 1935. [see pages 55 and 66]

[Han08] Yijie Han. Matching for graphs of bounded degree. In *Frontiers in Algorithmics*, volume 5059 of *Lecture Notes Comput. Sci.*, pages 171–173, 2008. [see page 66]

[HJL10] Bernhard Haeupler, Krishnam Raju Jampani, and Anna Lubiw. Testing simultaneous planarity when the common graph is 2-connected. In *Proceedings of the 21st Symposium on Algorithms and Computation (ISAAC'10)*, volume 6507 of *LNCS*, pages 410–421. Springer Heidelberg/Berlin, 2010. [see page 166]

[HJSW90] Derek A. Holton, Bill Jackson, Akira Saito, and Nicholas C. Wormald. Removable edges in 3-connected graphs. *Journal of Graph Theory*, 14(4):465–473, 1990. [see page 129]

[HKRS97] Monika R. Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55:3–23, 1997. [see pages 2 and 200]

[HN09] Seok-Hee Hong and Hiroshi Nagamochi. Two-page book embedding and clustered graph planarity. Tech. Report 2009-004, Department of Applied Mathematics & Physics, Kyoto University, 2009. [see pages 166, 189, and 190]

[Hsu02] Tsan-sheng Hsu. Simpler and faster biconnectivity augmentation. *J. Algorithms*, 45(1):55–71, 2002. [see page 24]

[HT73] John E. Hopcroft and Robert E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973. [see page 11]

[HT74] John E. Hopcroft and Robert E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974. [see pages v, 2, and 11]

[HT84] Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. on Computing*, 13(2):338–355, 1984. [see page 180]

[HT08] Bernhard Haeupler and Robert E. Tarjan. Planarity algorithms via pq-trees. *Electr. Notes Discrete Math.*, 31:143–149, 2008. [see pages 11 and 214]

[Huc93] Ulrich Huckenbeck. On paths in networks with valves. In *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science (STACS'93)*, volume 665 of *Lecture Notes Comput. Sci.*, pages 90–99, 1993. [see page 48]

[Huc97] Ulrich Huckenbeck. On valve adjustments that interrupt all s-t-paths in a digraph. *Journal of Automata, Languages and Combinatorics*, 2(1):19–45, 1997. [see page 48]

[HZ93] Pierre Hansen and Mao Lin Zheng. A linear algorithm for perfect matching in hexagonal systems. *Discrete Math.*, 122(1-3):179–196, 1993. [see page 66]

[JKR11] Vít Jelínek, Jan Kratochvíl, and Ignaz Rutter. A Kuratowski-type theorem for planariy of partially embedded graphs. In Ferran Hurtado and Marc van Kreveld, editors, *Proc. 27th Annu. Sympos. Comput. Geom.*, 2011. to appear. [see page 117]

[JM05] Martin Juvan and Bojan Mohar. 2-restricted extensions of partial embeddings of graphs. *European J. Comb.*, 26(3–4):339–375, 2005. [see pages 82 and 147]

[JS09] Michael Jünger and Michael Schulz. Intersection graphs in simultaneous embedding with fixed edges. *J. Graph Algorithms Appl.*, 13(2):205–218, 2009. [see pages 114, 118, 166, and 168]

[Kan93] Goos Kant. *Algorithms for Drawing Planar Graphs*. PhD thesis, University of Utrecht, 1993. [see page 25]

[Kan96] Goos Kant. Augmenting outerplanar graphs. *J. Algorithms*, 21(1):1–25, 1996. [see pages 24, 25, and 46]

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In J. W. Thatcher and R. E. Miller, editors, *Complexity of Computer Computations*. Plenum Press, 1972. [see page 18]

[KB91] Goos Kant and Hans L. Bodlaender. Planar graph augmentation problems. In Frank Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Proc. 2nd Workshop Algorithms and Data Structures (WADS'91)*, volume 519 of *Lecture Notes Comput. Sci.*, pages 286–298. Springer-Verlag, 1991. [see pages 23, 24, 25, and 26]

[KK03] Lukasz Kowalik and Maciej Kurowski. Short path queries in planar graphs in constant time. In *STOC '03*, pages 143–148, 2003. [see pages 87 and 106]

[KKRW10] Bastian Katz, Marcus Krug, Ignaz Rutter, and Alexander Wolff. Manhattan-geodesic embedding of planar graphs. In David Eppstein and Emden Gansner, editors, *Proc. 17th Internat. Sympos. Graph Drawing (GD'09)*, *Lect. Notes Comput. Sci.* Springer-Verlag, 2010. To appear. [see page 116]

[KN05] Sven O. Krumke and Hartmut Noltenmeier. *Graphentheoretische Konzepte und Algorithmen.* Teubner, 2005. [see page 38]

[KR92] Donald E. Knuth and Arvind Raghunathan. The problem of compatible representatives. *SIAM J. Discrete Math.*, 5(3):422–427, 1992. [see page 26]

[KR96] Claire Kenyon and Eric Rémila. Perfect matchings in the triangular lattice. *Discrete Math.*, 152(1–3):191–210, 1996. [see page 66]

[Kri00] Matthias Kriesell. Contractible subgraphs in 3-connected graphs. *J. Comb. Theory Ser. B*, 80:32–48, 2000. [see page 128]

[Kri02] Matthias Kriesell. A survey on contractible edges in graphs of a prescribed vertex connectivity. *Graphs and Combinatorics*, 18:1–30, 2002. [see page 128]

[KRW10] Bastian Katz, Ignaz Rutter, and Gerhard Woeginger. An algorithmic study of switch graphs. In *Proc. 35th International Workshop Graph-Theoretic Concepts in Computer Science (WG'09)*, volume 5911 of *Lecture Notes Comput. Sci.*, pages 226–237. Springer-Verlag, 2010. [see page 47]

[KS97] Jan Kratochvíl and Andras Sebő. Coloring precolored perfect graphs. *J. Graph Theory*, 25:207–215, 1997. [see page 82]

[Kur30] Kazimierz Kuratowski. Sur le problème des courbes gauches en topologie. *Fund. Math.*, 15:271–283, 1930. [see pages v, vii, 2, 11, and 117]

[KV08] Bernhard Korte and Jens Vygen. *Combinatorial Optimization.* Springer-Verlag, 4th edition edition, 2008. [see pages 7 and 52]

[KW02] Michael Kaufmann and Roland Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. Graph Algorithms Appl.*, 6(1):115–129, 2002. [see page 193]

[Lam70] Gerard Laman. On graphs and rigidity of plane skeletal structures. *J. Engineering Mathematics*, 4(4):331–340, 1970. [see page 63]

[Lic82] David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982. [see pages 18 and 26]

[LMPS92] Yanpei Liu, Paola Marchioro, Rosella Petreschi, and Bruno Simeone. Theoretical results on at most 1-bend embeddability of graphs. *Acta Math. Appl. Sinica (English Ser.)*, 8(2):188–192, 1992. [see page 192]

[LMS98] Yanpei Liu, Aurora Morgana, and Bruno Simeone. A linear algorithm for 2-bend embeddings of planar graphs in the two-dimensional grid. *Discrete Appl. Math.*, 81(1–3):69–91, 1998. [see page 192]

[LP86] László Lovász and Michael D. Plummer. *Matching Theory*. North Holland, Amsterdam, 1986. [see page 65]

[LT79] Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979. [see page 2]

[MdMS04] Aurora Morgana, Célia Picinin de Mello, and Giovanna Sontacchi. An algorithm for 1-bend embeddings of plane graphs in the two-dimensional grid. *Discrete Appl. Math.*, 141(1-3):225–241, 2004. Brazilian Symposium on Graphs, Algorithms and Combinatorics. [see page 193]

[Mei89] Christoph Meinel. Switching graphs and their complexity. In *Proceedings of the 14th Conference on Mathematical Foundations of Computer Science (MFCS'1989)*, volume 379 of *Lecture Notes Comput. Sci.*, pages 350–359. Springer-Verlag, 1989. [see page 47]

[Mel87] Avraham A. Melkman. On-line construction of the convex hull of a simple polyline. *Inform. Process. Lett.*, 25(1):11–12, 1987. [see pages 34 and 42]

[MMNS10] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. manuscript, 2010. [see page 115]

[MN95] Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995. [see pages 66, 199, and 214]

[Moh99] Bojan Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discr. Math.*, 12(1):6–26, 1999. [see page 82]

[MS04] Marcin Mucha and Piotr Sankowski. Maximum matchings via Gaussian elimination. In *Proc. 45th Annu. IEEE Sympos. Foundat. Comput. Sci. (FOCS'04)*, pages 248–255, 2004. [see page 65]

[MS06] Marcin Mucha and Piotr Sankowski. Maximum matchings in planar graphs via Gaussian elimination. *Algorithmica*, 45(1):3–20, 2006. [see pages 2, 3, and 65]

[Mut03] Petra Mutzel. The SPQR-tree data structure in graph drawing. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *ICALP '03*, volume 2719 of *Lecture Notes Comput. Sci.*, pages 34–46, 2003. [see page 82]

[MV80] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matchings in general graphs. In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci. (FOCS'80)*, pages 17–27, 1980. [see page 65]

[NB79] Takao Nishizeki and Ilker Baybars. Lower bounds on the cardinality of the maximum matchings of planar graphs. *Discrete Math.*, 28(3):255–267, 1979. [see pages vii, 4, 66, 67, and 76]

[Pat06] Maurizio Patrignani. On extending a partial straight-line drawing. *Found. Comput. Sci.*, 17(5):1061–1069, 2006. [see pages vii, 5, 82, and 116]

[PB99] J. Scott Provan and Roger C. Burk. Two-connected augmentation problems in planar graphs. *J. Algorithms*, 32:87–107, 1999. [see page 24]

[Pet91] Julius Petersen. Die Theorie der regulären Graphs. *Acta Mathematica*, 15:193–220, 1891. [see pages 65 and 66]

[Ple79] J. Plesńik. The NP-completeness of the Hamiltonian Cycle Problem in planar digraphs with degree bound two. *Information Processing Letters*, 8(4):199–201, 1979. [see pages 18 and 60]

[Pou94] Han La Poutré. Alpha-algorithms for incremental planarity testing. In *STOC '94*, pages 706–715, 1994. [see page 81]

[PW01] János Pach and Rephael Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17:717–728, 2001. [see page 10]

[PY81] Christos H. Papadimitriou and Mihalis Yannakakis. Worst-case ratios for planar graphs and the method of induction on faces. In *Proc. 22nd Annu. IEEE Sympos. Foundat. Comput. Sci. (FOCS'81)*, pages 358–363, 1981. [see page 66]

[Rap89] David Rappaport. Computing simple circuits from a set of line segments is NP-complete. *SIAM J. Comput.*, 18(6):1128–1139, 1989. [see pages 24, 25, and 26]

[Rei09] Klaus Reinhardt. The simple reachability problem in switch graphs. In *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'2009)*, volume 5404 of *Lecture Notes Comput. Sci.*, pages 461–472. Springer-Verlag, 2009. [see pages 48 and 49]

[RS04] Neil Robertson and Paul Seymour. Graph minors. XX. wagner's conjecture. *J. Combinat. Theor., Ser. B*, 92(2):325–357, 2004. [see page 163]

[RW08a] Ignaz Rutter and Alexander Wolff. Augmenting the connectivity of planar and geometric graphs. *Electr. Notes Discrete Math.*, 31:53–56, 2008. [see page 23]

[RW08b] Ignaz Rutter and Alexander Wolff. Augmenting the connectivity of planar and geometric graphs. Technical Report 2008-3, Universität Karlsruhe, 2008. Available at `www.ubka.uni-karlsruhe.de/indexer-vvv/ira/2008/3`. [see page 23]

[RW08c] Ignaz Rutter and Alexander Wolff. Computing large matchings fast. In *Proc. 19th Annu. ACM-SIAM Sympos. Discr. Algorithms (SODA'08)*, pages 183–192, 2008. [see page 66]

[RW10] Ignaz Rutter and Alexander Wolff. Computing large matchings fast. *ACM Trans. Algorithms*, 1, 2010. [see page 66]

[Sch99] Alexander Schrijver. Bipartite edge coloring in $O(\Delta m)$ time. *SIAM J. Comput.*, 28:841–846, 1999. [see page 66]

[SGYBD05] Roded Sharan, Jens Gramm, Zohar Yakhini, and Amir Ben-Dor. Multiplexing schemes for generic SNP genotyping assays. *Journal of Comp. Biology*, 15:514–533, 2005. [see page 48]

[SLL07] Jeremiy Siek, Lie-Quan Lee, and Andrew Lumsdaine. The Boost Graph Library documentation. `www.boost.org/libs/graph`, visited 07/04/2007. [see page 65]

[Syl78] John J. Sylvester. Chemstry and algebra. *Nature*, 17:284, 1878. [see page 1]

[Tam87] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987. [see pages 10, 193, 194, 195, 196, and 199]

[Tam96] Roberto Tamassia. On-line planar graph embedding. *J. Algorithms*, 21(2):201–239, 1996. [see page 81]

[Tam98] Roberto Tamassia. Constraints in graph drawing algorithms. *Constraints*, 3(1):87–120, 1998. [see page 82]

[Tar72] Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 2:146–160, 1972. [see page 87]

[Tar83] Robert E. Tarjan. *Data structures and network algorithms.* SIAM, Philadelphia, 1983. [see pages 58, 65, and 75]

[TDB88] Roberto Tamassia, Giuseppe Di Battista, and Carlo Batini. Automatic graph drawing and readability of diagrams. *IEEE Trans. Syst., Man and Cyber.*, 18(1):61–79, 1988. [see page 82]

[Thu90] William P. Thurston. Conway's tiling groups. *Amer. Math. Monthly*, 97(8):757–773, 1990. [see page 66]

[Tót08] Csaba D. Tóth. Connectivity augmentation in plane straight line graphs. *Electronic Notes in Discrete Mathematics*, 31:49–52, 2008. [see pages 24 and 39]

[Tut47] William T. Tutte. The factorization of linear graphs. *J. Lond. Math. Soc.*, 22:107–111, 1947. [see page 66]

[TV09] Csaba D. Tóth and Pavel Valtr. Augmenting the edge connectivity of planar straight line graphs to three. In *Proc. XIII Encuentros de Geometría Computacional (EGC'09)*, Zaragoza, 2009. [see page 24]

[Wes92] Jeffery Westbrook. Fast incremental planarity testing. In W. Kuich, editor, *ICALP '92*, volume 623 of *Lecture Notes Comput. Sci.*, pages 342–353, 1992. [see page 81]

[Whi32] Hassler Whitney. Congruent graphs and the connectivity of graphs. *American J. Math.*, 54(1):150–168, 1932. [see pages 10 and 27]

[YZ07] Raphael Yuster and Uri Zwick. Maximum matching in graphs with an exluded minor. In *Proc. 18th Annu. ACM-SIAM Sympos. Discr. Algorithms (SODA'07)*, pages 108–117, 2007. [see page 65]

List of Publications

Journal Articles

[1] **Computing Large Matchings in Planar Graphs with Fixed Minimum Degree**. *Theoretical Computer Science*, 412(32):4092–4099, 2010. Joint work with Robert Franke and Dorothea Wagner.

[2] **Computing Large Matchings Fast**. *ACM Transactions on Algorithms*, 7(1), 2010. Joint work with Alexander Wolff.

[3] **Augmenting the Connectivity of Planar and Geometric Graphs**. *Electronic Notes in Discrete Mathematics*, 31:53–56, 2008. Joint work with Alexander Wolff.

Articles in Refereed Conference Proceedings

[4] **Generalizing Geometric Graphs**. In: *Proceedings of the 19th International Symposium on Graph Drawing (GD'11)*, Lecture Notes in Computer Science. Springer, 2012. To appear, joint work with Edith Brunel, Andreas Gemsa, Marcus Krug, and Dorothea Wagner.

[5] **Hamiltonian Orthogeodesic Alternating Paths**. In: *Proceedings of the 22nd International Workshop on Combinatorial Algorithms*, Lecture Notes in Computer Science. Springer, 2012. To appear, joint work with Emilio Di Giacomo, Luca Grilli, Marcus Krug, and Giuseppe Liotta.

[6] **Connecting Two Trees with Optimal Routing Cost**. In: *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG '11)*. University of Toronto Press, 2011. To appear, joint work with Mong-Jen Kao, Bastian Katz, Marcus Krug, D.T. Lee, Martin Nöllenburg, and Dorothea Wagner.

[7] **Sliding Labels for Dynamic Point Labeling**. In: *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG '11)*. University of Toronto Press, 2011. To appear, joint work with Andreas Gemsa and Martin Nöllenburg.

[8] **The Density Maximization Problem in Graphs**. In: *Proceedings of the 17th Annual International Conference on Computing Combinatorics (COCOON'11)*, Lecture Notes in Computer Science. Springer, 2011. To appear, joint work with Mong-Jen Kao, Bastian Katz, Marcus Krug, D.T. Lee, and Dorothea Wagner.

[9] **Consistent Labeling of Rotating Maps**. In: *Algorithms and Data Structures, 12th International Symposium (WADS'11)*, volume 6844 of *Lecture Notes in Computer Science*, pages 451–462. Springer, August 2011. Full version available at `http://arxiv.org/abs/1104.5634`., Joint work with Andreas Gemsa and Martin Nöllenburg.

[10] **A Kuratowski-Type Theorem for Planarity of Partially Embedded Graphs** . In: *Proceedings of the 27th Annual ACM Symposium on Computational Geometry (SoCG'11)*, pages 107–116. ACM Press, 2011. Joint work with Vít Jelínek and Jan Kratochvíl.

[11] **Speed Dating: An Algorithmic Case Study Involving Matching and Scheduling**. In: *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 2011. Joint work with Bastian Katz, Ben Strasser, and Dorothea Wagner.

[12] **On d-regular Schematization of Embedded Paths**. In: *Proceedings of the 37th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'11)*, volume 6543 of *Lecture Notes in Computer Science*, pages 260–271. Springer, January 2011. Joint work with Andreas Gemsa, Martin Nöllenburg, and Thomas Pajor.

[13] **Orthogonal Graph Drawing with Flexibility Constraints**. In: *Proceedings of the 18th International Symposium on Graph Drawing (GD'10)*, volume 6502 of *Lecture Notes in Computer Science*, pages 92–104. Springer, 2011. Joint work with Thomas Bläsius, Marcus Krug, and Dorothea Wagner.

[14] **Testing the Simultaneous Embeddability of Two Graphs whose Intersection is a Biconnected Graph or a Tree**. In: *Proceedings of the 21st International Workshop on Combinatorial Algorithms*, volume 6460 of *Lecture Notes in Computer Science*, pages 212–225. Springer, 2011. Joint work with Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, and Maurizio Patrignani.

[15] **Testing Planarity of Partially Embedded Graphs**. In: *Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 202–221. SIAM, 2010. Joint work with Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, and Maurizio Patrignani.

[16] **Gateway Decompositions for Constrained Reachability Problems**. In: *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*, pages 449–461. Springer, May 2010. Joint work with Bastian Katz, Marcus Krug, Andreas Lochbihler, Gregor Snelting, and Dorothea Wagner.

[17] **Manhattan-Geodesic Embedding of Planar Graphs**. In: *Proceedings of the 17th International Symposium on Graph Drawing (GD'09)*, volume 5849 of *Lecture Notes in Computer Science*, pages 207–218. Springer, 2010. Joint work with Bastian Katz, Marcus Krug, and Alexander Wolff.

[18] **Computing Large Matchings in Planar Graphs with Fixed Minimum Degree**. In: *Proceedings of the 20th International Symposium on Algorithms and*

Computation (ISAAC'09), volume 5878 of *Lecture Notes in Computer Science*, pages 872–881. Springer, 2009. Joint work with Robert Franke and Dorothea Wagner.

[19] **An Algorithmic Study of Switch Graphs**. In: *Proceedings of the 35th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'09)*, volume 5911 of *Lecture Notes in Computer Science*, pages 226–237. Springer, June 2009. Joint work with Bastian Katz and Gerhard J. Woeginger.

[20] **Augmenting the connectivity of planar and geometric graphs**. In: *Topological & Geometric Graph Theory (TGGT'08)*, pages 55–58, 2008. Joint work with Alexander Wolff.

[21] **Computing Large Matchings Fast**. In: *Proceedings of the 19th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 183–192, 2008. Joint work with Alexander Wolff.

Articles in Non-Refereed Workshop Proceedings

[22] **How Alexander the Great Brought the Greeks Together While Inflicting Minimal Damage to the Barbarians**. In: *Proceedings of the 26th European Workshop on Computational Geometry (EuroCG'10)*, pages 73–76, 2010. Joint work with Mark de Berg, Dirk Gerrits, Amirali Khosravi, Constantinos Tsirogiannis, and Alexander Wolff.

Thesis

[23] **Schnelle Berechnung von großen Matchings**. Master's thesis, Fakultät für Informatik, Universität Karlsruhe, April 2007.

Posters

[24] **Automatic Generation of Route Sketches**. In: *Proceedings of the 18th International Symposium on Graph Drawing (GD'10)*, volume 6502 of *Lecture Notes in Computer Science*, pages 391–392. Springer, 2011. Poster abstract., Joint work with Andreas Gemsa, Martin Nöllenburg, and Thomas Pajor.

Curriculum Vitæ

Name	Ignaz Rutter
Date of Birth	13 May 1981
Place of Birth	Karlsruhe
Nationality	German

06/2000	Abitur (university entrance qualification), Copernicus-Gymnasium Philippsburg
10/2000–08/2001	Alternative civilian service, as youth worker at the German Young Christian Workers association (CAJ)
10/2001–04/2007	Student in Informatics at Universität Karlsruhe (TH). Finished with Dilpoma in Informatics.
05/2007–04/2008	Ph.D. student and research assistent in the project "Geometric Networks and their Visualization" funded by the German Research Foundation (DFG), Fakultät für Informatik, Universität Karlsruhe (TH). Advisor: Prof. Dr. Alexander Wolff
since 05/2008	Ph.D. student and reasearch assistent, chair *Algorithmics I*, Karlsruhe Institute of Technology (KIT). Advisor: Prof. Dr. Dorothea Wagner
05.04–10.04.2009	Research guest of Prof. Jan Kratochvíl at Charles University, Prague, Czech Republic
12.10–16.10.2009	Research guest of Prof. Giuseppe Di Battista and Prof. Maurizio Patrignani at University Roma Tré, Rome, Italy