Schnelle Berechnung von großen Matchings

Diplomarbeit am Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner
Fakultät für Informatik
Universität Karlsruhe (TH)

von

Ignaz Rutter

Erstgutachter:	PD Dr. habil. Alexander Wolff
Zweitgutachter:	Prof. Dr. Dorothea Wagner

März 2007

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt zu haben. Die verwendete Literatur und sonstige Hilfsmittel sind vollständig angegeben.

Karlsruhe, 30. 3. 2007

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Ein *Matching* (oder eine *Zuordnung*) in einem ungerichteten Graphen $G = (V, E)$ ist eine Kantenmenge M mit der Eigenschaft, dass keine zwei Kanten aus M zu einem gemeinsamen Knoten inzident sind. Die Suche nach Matchings in Graphen hat viele Anwendungen in der Informatik und im Operations Research. Beispielsweise führt folgende Aufgabenstellung zu einem Matching-Problem in einem bipartiten Graphen: Gegeben seien eine Menge von Arbeitern und eine Menge von Aufgaben. Zudem sei bekannt, welcher Arbeiter welche Aufgaben erledigen kann. Gesucht ist eine Zuordnung von Arbeitern zu Aufgaben, so dass möglichst viele Aufgaben gleichzeitig abgearbeitet werden können. Modelliert man jeden Arbeiter und jede Aufgabe als Knoten eines Graphen und verbindet einen Arbeiter mit einer Aufgabe, wenn der Arbeiter diese Aufgabe erfüllen kann, so entspricht eine Zuordnung, bei der eine maximale Anzahl von Aufgaben gleichzeitig abgearbeitet werden kann, genau einem größten Matching in diesem Graphen.

Ein ähnliches Beispiel führt zu einer Anwendung von Matchings in allgemeinen Graphen. Eine Firma hat n Arbeiter und es sollen Zweiergruppen gebildet werden. Allerdings kann nicht jedes Paar von Arbeitern gut zusammen arbeiten. Es soll eine möglichst große Anzahl von Teams gefunden werden, die gut zusammen arbeiten können. Zur Lösung betrachtet man den Graphen, der für jeden Arbeiter einen Knoten besitzt. Zwei Knoten werden genau dann durch eine Kante verbunden, wenn die entsprechenden Arbeiter gut zusammen arbeiten können. Eine Zuteilung, bei der eine maximale Anzahl von Pärchen gebildet wird, entspricht genau einem größten Matching in diesem Graphen.

Alle in dieser Arbeit vorkommenden Graphen sind, soweit nichts anderes vorausgesetzt wird, zusammenhängend und schlicht, enthalten also keine Schlingen und Mehrfachkanten. Beides stellt keine wesentliche Einschränkung dar, da ein Matching in einem nicht zusammenhängenden Graphen eine Vereinigung von Matchings der Einzelkomponenten ist. Mehrfache Kanten und Schlingen verändern die Größe eines Matchings nicht, sie können also weggelassen werden.

Es gibt eine ganze Reihe schon klassischer Resultate. Der erste Polynomialzeitalgorithmus für größte Matchings in allgemeinen Graphen stammt von Edmonds [Edm65] und hat eine Laufzeit von $O(|V|^4)$. Für Matchings in bipartiten Graphen gaben Hopcroft und Karp [HK73] einen Algorithmus an, der in $O(\sqrt{n}m)$ Zeit ein Matching maximaler Kardinalität findet. Dabei ist $n = |V|$ und $m = |E|$. Sieben Jahre später konnten Micali und Vazirani zeigen, dass auch in allgemeinen Graphen ein Matching maximaler Kardinalität in $O(\sqrt{n}m)$ Zeit gefunden werden kann [MV80]. Derzeit sind keine schnelleren Algorithmen für diese Probleme bekannt. Der Schnelle Algorithmus Für

Allgemeine Graphen ist deutlich komplizierter als der für den bipartiten Fall. Die Algorithmenbibliothek LEDA bietet beispielsweise eine Implementierung des Algorithmus für bipartite Graphen. Für Matchings in allgemeinen Graphen wird allerdings ein einfacheres Verfahren mit Laufzeit $O(nm\alpha(n,m))$ verwendet [Alg07]. Dabei bezeichnet $\alpha(n,m)$ die Inverse Ackermann-Funktion. Diese wächst so langsam, dass sie für alle realistischen Eingabegrößen Werte kleiner oder gleich 4 annimmt [CLRS01].

Es wurde intensiv untersucht, wann ein Graph ein perfektes Matching (also ein Matching der Größe $n/2$) hat. Es ist bekannt, dass alle zweifach zusammenhängenden 3-regulären Graphen und alle bipartiten k-regulären Graphen ein perfektes Matching besitzen [Pet91, Kőn16]. Für diese Fälle existieren schnelle Algorithmen, um ein perfektes Matching zu finden [BBDL01, Sch99]. Schnell bedeutet hierbei, dass sie ein solches Matching schneller finden als der Algorithmus zum Finden eines größten Matchings in allgemeinen Graphen.

Tutte zeigte notwendige und hinreichende Bedingungen für die Existenz eines perfekten Matchings in beliebigen Graphen [Tut47]. Leider sind keine Algorithmen bekannt, die ein solches perfektes Matching schneller berechnen können als ein Matching maximaler Kardinalität.

Ist das perfekte Matching eines Graphen jedoch eindeutig, so kann es auch schnell gefunden werden. Gabow et al. [GKT99] stellen einen Algorithmus vor, der in $O(m \log^4 n)$ Zeit überprüft, ob ein Graph ein eindeutiges perfektes Matching besitzt. Ist das perfekte Matching eindeutig, so wird es gleichzeitig konstruiert. Ist der Graph planar, so ist dafür sogar nur $O(n \log n)$ Zeit nötig.

Neben diesen algorithmischen Fragestellungen gibt es eine Reihe theoretischer Resultate über die Existenz von Matchings einer gewissen Mindestgröße. Nishizeki und Baybars [NB79] geben Schranken für die Größe von Matchings in planaren Graphen in Abhängigkeit von Minimalgrad und Zusammenhang an. Biedl et al. [BDD+04] verbesserten einige der Ergebnisse von Nishizeki und Baybars und gaben eine Reihe weiterer Schranken in Abhängigkeit von Maximalgrad, Zusammenhang und Planarität an. Diese Schranken bilden den Ausgangspunkt für diese Diplomarbeit. Sie sind in Tabelle 1.1 dargestellt. Dort bezeichnen ℓ_2 und ℓ_4 die Anzahlen der Blätter des 2- bzw. 4-Blockbaums (also des Baum der 2- bzw. 4-fach-Zusammenhangskomponenten). Schranke 1 geht dabei aus Schranke 2 hervor, indem Schranken für die Werte von ℓ_2 bzw. ℓ_4 angegeben werden.

Die Beweise für diese Schranken sind reine Existenzbeweise, d.h. nicht konstruktiv. Sie verwenden einen Satz von Berge [Ber57]:

Satz 1 (Berge)**.** *Sei $G = (V, E)$ ein Graph. Ist $V' \subseteq V$, so bezeichnet* $\mathrm{odd}(V')$ *die Anzahl der Komponenten von $G - V'$ mit ungerader Knotenzahl. Für jedes $V' \subseteq V$ und jedes Matching M ist die Anzahl der ungematchten Knoten in M mindestens* $\mathrm{odd}(V') - |V'|$*. Ferner existiert eine Menge $V' \subseteq V$, so dass jedes Matching mit maximaler Kardinalität genau* $\mathrm{odd}(V') - |V'|$ *ungematchte Knoten hat.*

Die Mindestgrößen der Matchings von Nishizeki und Baybars [NB79] und Biedl et al. [BDD+04] ergeben sich dann durch obere Schranken von $\mathrm{odd}(V') - |V'|$.

Graph	Schranke 1	Schranke 2
dreifach zusammenhängend, planar	$\frac{n+4}{3}$	$\frac{2n+4-\ell_4}{4}$
Maximalgrad 3	$\frac{n-1}{3}$	$\frac{3n-n_2-2\ell_2}{6}$
Maximalgrad k	$\frac{m}{2k-1}$	
3-regulär	$\frac{4n-1}{9}$	$\frac{3n-2\ell_2}{6}$

Tabelle 1.1: Schranken für die Existenz von Matchings

Diese Arbeit beschäftigt sich mit der Fragestellung, ob große Matchings, von denen man weiß, dass sie existieren, schneller gefunden werden können, als durch Berechnung eines Matchings größter Kardinalität. Es werden alle Schranken in Tabelle 1.1 bis auf Schranke 2 in dreifach zusammenhängenden planaren Graphen erreicht. Das heißt, für alle diese Fälle werden Algorithmen angegeben, die Matchings von mindestens dieser Größe schneller finden als der schnellste bekannte Algorithmus zur Bestimmung von Matchings maximaler Kardinalität. Tabelle 1.2 bietet einen Überblick über die Resultate dieser Arbeit. Sie ist wie folgt strukturiert:

In Kapitel 2 werden zunächst einige Begriffe eingeführt und grundlegende Aussagen vorgestellt.

In Kapitel 3 wird ein Algorithmus angegeben, der in einem Baum ein Matching maximaler Kardinalität in linearer Zeit berechnet. Außerdem wird eine untere Schranke für die Größe von Matchings maximaler Kardinalität in Bäumen mit beschränktem Maximalgrad angegeben. In Graphen mit Maximalgrad 3 lässt sich damit bereits ein Matching der Größe $(n-1)/3$ finden, was einer der Schranken von Biedl et al. [BDD+04] entspricht.

In Kapitel 4 wird die Existenz großer Matchings in Graphen mit beschränktem Maximalgrad untersucht. Es wird gezeigt, dass in jedem solchen Graphen ein Matching der Größe $(m-1)/(2k-2)$, in planaren Graphen sogar der Größe $m/(k+4)$ existiert. Zusätzlich werden Linearzeitalgorithmen angegeben, die solche Matchings finden.

Kapitel 5 untersucht, inwiefern ein stärkerer Zusammenhang in Graphen mit beschränktem Maximalgrad hilft, größere Matchings zu finden.

Kapitel 6 beschäftigt sich mit Graphen mit Maximalgrad 3. In solchen Graphen existiert stets ein Matching der Größe $(3n-n_2-2\ell_2)/6$ [BDD+04]. Es wird ein Algorithmus angegeben, der ein solches Matching in $O(n \log^4 n)$ Zeit berechnet. Zudem wird gezeigt, dass diese Schranke auch dann noch scharf ist, wenn zirka ein Drittel aller Knoten Grad 2 hat. Dies beantwortet eine offene Frage von Biedl et al. [BDD+04]. Sie gaben zwar eine Familie von Graphen an, für die die

Graphenklasse	Schranke für Matchinggröße		Laufzeit
	Schranke 1	Schranke 2	$O(\cdot)$
3-regulär	$(4n-1)/9$	$(3n-2\ell_2)/6$	$n \log^4 n$
Maximalgrad 3	$(n-1)/3$	$(3n-n_2-2\ell_2)/6$	$n \mid n \log^4 n$
$\kappa(G) \geq 3$, planar	$(n+4)/3$	$(2n+4-6\ell_4)/4$	$n\,\alpha(n)$
trianguliert, planar	$(2n+4-2\ell_4)/4$		$n\,\alpha(n)$
Maxgrad k	$(n-1)/k$		n
Maxgrad k	$(m-1)/(2k-2)$		n
Maxgrad k, $\kappa(G) \geq 2$	$2(n-1)/(k+3)$		$n \log^4 n$
Maxgrad k, $\kappa(G) \geq 2$, planar	$2(n-1)/(k+3)$		$n \log^2 n$
Graphen mit gradbeschränktem 2- bzw. 4-Blockbaum			
3-regulär	größtes Matching		$n \log^4 n$
3-regulär, planar	größtes Matching		$n\,\alpha(n)$
$\kappa(G) \geq 3$, planar	größtes Matching		$n\,\alpha(n)$

Tabelle 1.2: Übersicht über die Resultate dieser Arbeit. Dabei bezeichnet κ den Knotenzusammenhang, ℓ_2 die Anzahl der Blätter des 2-Blockbaums und ℓ_4 die Anzahl der Blätter des 4-Blockbaums. Sind zwei durch | getrennte Laufzeiten angeben, so bezieht sich die linke Angabe auf den Algorithmus zum Erreichen von Schranke 1, die rechte Angabe auf Schranke 2.

Schranke scharf ist, diese enthielt aber keine Knoten mit Grad 2. Sie fragten, ob es möglich ist ein Beispiel mit signifikanter Anzahl von Grad-2-Knoten zu konstruieren, das die Schranke erreicht.

In Kapitel 7 geht es darum, wie in vierfach zusammenhängenden planaren Graphen stets ein (fast) perfektes Matching in linearer Zeit gefunden werden kann. Die Darstellung basiert auf einer Arbeit von Chiba und Nishizeki [CN89].

In Kapitel 8 wird zunächst gezeigt, wie in dreifach zusammenhängenden planaren Graphen ein Matching der Größe $(n-1)/3$ in Linearzeit gefunden werden kann. Für Graphen, deren 4-Blockbaum ein Pfad ist, wird ein Verfahren angegeben, das ein (fast) perfektes Matching in $O(n\alpha(n))$ Zeit findet. Dieses Resultat wird anschließend verwendet, um mit einer ähnlichen Technik wie in Kapitel 6 in dreifach zusammenhängenden planaren Graphen ein Matching der Größe $(2n+4-6\ell_4)/4$ zu finden. Dabei bezeichnet ℓ_4 die Anzahl der Blätter im 4-Blockbaum. Damit bleibt das Ergebnis des Algorithmus etwas schlechter als die Schranke $(2n+4-\ell_4)/4$ von Biedl et al. [BDD+04]. In trianguierten Graphen garantiert der Algorithmus eine etwas bessere Schranke, nämlich $(2n+4-2\ell_4)/4$.

Wie bereits erwähnt existiert in 3-regulären Graphen, deren 2-Blockbaum ein Pfad ist, stets ein perfektes Matching [Pet91]. Dieses kann in $O(n \log^4 n)$ Zeit gefunden werden [BBDL01]. In Kapitel 9 wird gezeigt, wie in 3-regulären Graphen, deren 2-Blockbaum konstanten Maximalgrad besitzt, ein Matching maximaler Kardinalität schnell gefunden werden kann. Die benötigte Zeit ist $O(n \log^4 n)$, in planaren Graphen sogar nur $O(n\alpha(n))$. Ähnlich kann auch in dreifach zusammenhängenden planaren Graphen, deren 4-Blockbaum konstan-

ten Maximalgrad besitzt, ein Matching maximaler Kardinalität in $O(n\alpha(n))$ Zeit gefunden werden. Es ist unwahrscheinlich, dass diese beiden Resultate noch verallgemeinert werden können, da Biedl [Bie01] zeigt, dass das Finden größter Matchings in allgemeinen Graphen in Linearzeit auf das Finden größter Matchings in 3-regulären Graphen reduziert werden kann. Wäre es also möglich zusätzlich noch die Gradbeschränkung des Blockbaums entfallen zu lassen, so ergäbe sich ein schnellerer Algorithmus für Matchings in allgemeinen Graphen. In ähnlicher Weise kann das Problem des Findens größter Matchings in planaren Graphen in Linearzeit auf das Finden größter Matchings in planaren, triangulierten Graphen mit Maximalgrad 9 (also speziellen dreifach zusammenhängenden planaren Graphen) reduziert werden.

In Kapitel 10 werden die Ergebnisse der Arbeit zusammengefasst und einige offene Fragen zur weiteren Untersuchung gestellt.

2 Terminologie und wichtige Grundlagen

Ein *Graph* ist ein Tupel $G = (V, E)$, dabei ist V die Menge der Knoten und $E \subseteq V \times V$ die Menge der Kanten. In ungerichteten Graphen werden die Kanten (u, v) und (v, u) identifiziert und als ungeordnetes Paar $\{u, v\}$ geschrieben. Eine Kante $e = \{u, v\}$ und ein Knoten w heißen *inzident*, wenn $w = u$ oder $w = v$ gilt. Oft wird statt $\{u, v\}$ auch einfach uv geschrieben. Zwei Knoten u und v heißen *adjazent*, wenn die Kante $\{u, v\}$ in E enthalten ist. Der *Grad* eines Knoten v ist die Anzahl der zu v inzidenten Kanten, er wird mit $\deg_G(v)$ oder auch mit $\deg(v)$ bezeichnet, wenn der Graph G aus dem Kontext klar ist.

Ein Graph heißt *bipartit*, wenn sich seine Knoten in zwei Mengen partitionieren lassen, so dass keine Kante zwischen zwei Knoten in derselben Menge existiert.

Ist $V' \subseteq V(G)$, so bezeichnet $G - V'$ den Graphen, der durch Entfernen aller Knoten von V' aus $V(G)$ samt ihrer inzidenten Kanten entsteht. Sind $u, v \in V(G)$, so bezeichnet $G + uv$ den Graphen $G' = (V(G), E(G) \cup \{uv\})$.

2.1 Nicht erweiterbare, größte und (fast) perfekte Matchings

Ein *Matching* ist eine Menge von Kanten M mit der Eigenschaft, dass keine zwei Kanten aus M zu demselben Knoten inzident sind. Man ist insbesondere an großen Matchings interessiert.

Im Zusammenhang mit Matchings gibt es einige wichtige Begriffe. Sei $G = (V, E)$ ein Graph und $M \subseteq E$ ein Matching. Ein Knoten $v \in V(G)$ heißt *gematcht (bezüglich M)*, wenn es eine Kante in M gibt, die zu v inzident ist. Andernfalls heißt v *frei (bezüglich M)*.

Ein *perfektes Matching* ist ein Matching, bei dem alle Knoten gematcht sind. Ein perfektes Matching kann nur dann existieren, wenn der Graph eine gerade Knotenanzahl besitzt. Ist dies nicht der Fall, so muss immer mindestens ein Knoten frei bleiben. Dann existiert bestenfalls ein *fast perfektes Matching*: Ein Matching, bezüglich dem genau ein Knoten frei ist.

Leider gibt es nicht in jedem Graphen ein perfektes oder fast perfektes Matching. Daher ist man an möglichst großen Matchings interessiert. Dabei gibt es zwei Arten von besonders großen Matchings: *Nicht erweiterbare* Matchings (engl. *maximal* matchings) und Matchings *maximaler Kardinalität* (engl. *maximum* matchings).

Ein *nicht erweiterbares Matching* ist ein Matching, das nicht durch Hinzunahme von Kanten vergrößert werden kann. Es ist also bezüglich Inklusion maximal. Ein *größtes Matching* hingegen ist ein Matching maximaler Kardinalität. Ein größtes Matching ist nicht erweiterbar, die Umkehrung gilt aber nicht. Abbildung 2.1 zeigt ein Beispiel für ein nicht erweiterbares Matching, das kein größtes Matching ist. Die Kanten der Matchings sind jeweils hervorgehoben.

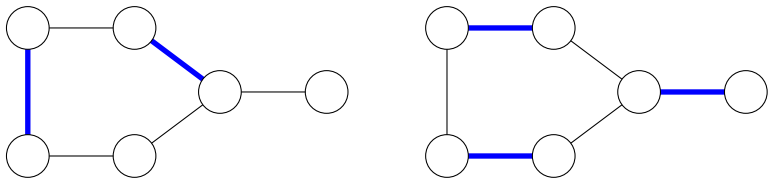


Abbildung 2.1: Graph mit einem nicht erweiterbaren Matching (a) und einem größten Matching (b).

Nicht erweiterbare Matchings sind vor allem interessant, weil sie mit einem Greedy-Verfahren sehr schnell berechnet werden können. Um trotzdem eine gewisse Mindestgröße garantieren zu können, ist man an unteren Schranken für die Größe von nicht erweiterbaren Matchings interessiert. Dabei hilft folgendes Lemma:

Lemma 1 ([BDD+04])**.** *Besitzt ein Graph G ein größtes Matching der Größe k, so hat jedes nicht erweiterbare Matching mindestens Größe $k/2$.*

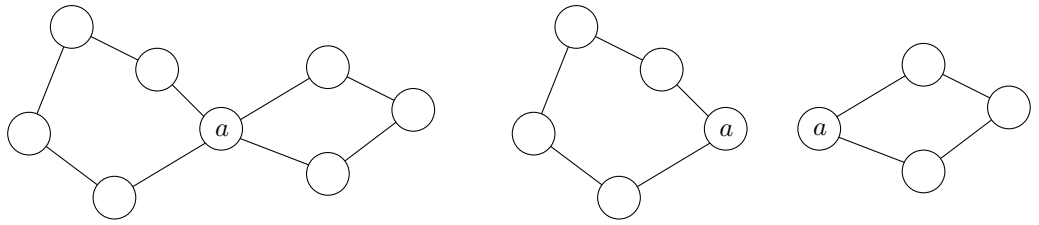
Beweis. Sei M ein größtes Matching in G und $|M| = k$. Weiter sei M' ein nicht erweiterbares Matching in G. Für jede Kante e in M muss mindestens einer der Endpunkte auch in M' gematcht sein. Andernfalls sei e eine Kante, deren Endpunkte nicht gematcht sind, dann könnte e aber zu M' hinzugefügt werden. Daher sind mindestens k Knoten von G bezüglich M' gematcht. Folglich gilt $|M'| \geq k/2$. □

Wann immer also eine untere Schranke für die Größe eines größten Matchings angegeben wird, ergibt diese nach Division durch 2 eine untere Schranke für die Größe eines nicht erweiterbaren Matchings.

2.2 Planare Graphen

Ein Graph G heißt planar, wenn er sich ohne Kantenkreuzung in die euklidische Ebene zeichnen lässt. Als *Einbettung* bezeichnet man die Information für jeden Knoten, in welcher Reihenfolge die Kanten im Uhrzeigersinn ausgehen, sowie welche Facette die Äußere ist. Für einen planaren Graphen kann in $O(n)$ Zeit eine solche Einbettung bestimmt werden [HT74].

Eine Zeichnung von G unterteilt die Ebene in zusammenhängende Bereiche, die *Facetten*, die durch Kanten des Graphen begrenzt werden. Diese Facetten sind nur von der Einbettung abhängig, nicht von der konkreten Zeichnung.



(a) (b)

Abbildung 2.2: Zusammenhängender Graph, der nicht zweifach zusammenhängend ist (a) und seine Zweifachzusammenhangskomponenten (b).

Für die Anzahl der Knoten n, Kanten m und Facetten f gilt die eulersche Polyederformel: $n - m + f = 2$.

Insbesondere folgt daraus, dass ein planarer Graph höchstens $3n - 6$ Kanten besitzt, also für einen zusammenhängenden planaren Graphen $m = \Theta(n)$ gilt.

2.3 Zusammenhang und Blockbäume

Ein weitere Eigenschaften von Graphen ist der Zusammenhang. Ein Graph heißt einfach zusammenhängend, wenn er zusammenhängend ist. Ein Graph ist d-fach zusammenhängend, wenn er $(d-1)$-fach zusammenhängend ist und $G - T$ für jedes $T \subseteq V(G)$ der Kardinalität $|T| = d - 1$ zusammenhängend ist. Ein d-fach zusammenhängender Graph zerfällt also nicht in mehrere Komponenten, wenn man weniger als d beliebige Knoten entfernt. Abbildung 2.2 (a) zeigt ein Beispiel für einen zusammenhängenden Graphen, der nicht zweifach zusammenhängend ist.

Ist ein zusammenhängender Graph nicht zweifach zusammenhängend, so existiert ein Knoten a, so dass $G - \{a\}$ mehrere Komponenten besitzt. In diesem Fall heißt a *Separatorknoten*. Eine Zweifachzusammenhangskomponente ist ein maximaler zweifach zusammenhängender Teilgraph. Abbildung 2.2 zeigt die Zerlegung eines Graphen in seine Zweifachzusammenhangskomponenten. Separatorknoten sind in mehreren Komponenten enthalten.

Der 2-*Blockbaum* eines zusammenhängenden Graphen G ist ein Graph, der für jeden Separatorknoten von G und jede Zweifachzusammenhangskomponente von G einen Knoten enthält. Ist ein Separatorknoten in einer Zweifachzusammenhangskomponente enthalten, so werden die entsprechenden Knoten im 2-Blockbaum durch eine Kante verbunden. Das definiert einen Graphen ohne Kreise, denn lägen einige Komponenten auf einem Kreis, so würden sie zu einer einzigen großen Zweifachzusammenhangskomponente gehören. Der 2-Blockbaum von G wird auch mit $\mathbb{BCT}(G)$ (engl. block-cut-tree) bezeichnet. Die Anzahl der Blätter dieses Baumes wird mit $\ell_2(G)$ bezeichnet. Ist der Graph aus dem Kontext klar, so wird er oft auch weggelassen. Der 2-Blockbaum beschreibt, wie die Zweifachzusammenhangskomponenten miteinander verbunden sind, und gibt daher in gewisser Weise die Grobstruktur des Graphen wider.

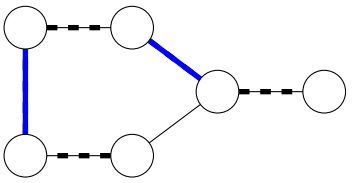


Abbildung 2.3: Augmentierender Pfad.

Eine ähnliche Definition ist auch für dreifach zusammenhängende Graphen möglich. Ist ein solcher Graph nicht vierfach zusammenhängend, so existieren drei Knoten u, v, w, so dass $G - \{u, v, w\}$ mehrere Komponenten besitzt. Das Tripel $\{u, v, w\}$ heißt dann *separierendes Tripel*. Ein separierendes Tripel kann verwendet werden, um einen Graphen zu zerlegen: Für jede Komponente C von $G - \{u, v, w\}$ wird ein neuer Graph gebildet, indem zu C die Knoten u, v, w hinzugefügt werden, und zwar zusammen mit all ihren Kanten, die zu einem anderen Knoten in C inzident sind. Zudem werden noch die Kanten uv, vw, uw hinzugefügt, sofern sie noch nicht vorhanden sind.

Dieser Prozess wird iteriert, bis alle entstanden Graphen vierfach zusammenhängend sind. Das sind die Vierfachzusammenhangskomponenten von G. Der 4-*Blockbaum* ist dann ein Graph, der für jedes separierende Tripel und für jede Vierfachzusammenhangskomponente einen Knoten enthält. Die Knoten eines separierendes Tripels und einer Vierfachzusammenhangskomponenten sind genau dann miteinander verbunden, wenn das separierende Tripel in der Vierfachzusammenhangskomponenten enthalten ist. Dieser Graph ist wieder ein Baum, und die Anzahl seiner Blätter wird mit $\ell_4(G)$, oder auch nur mit ℓ_4 bezeichnet, wenn der Graph G aus dem Kontext klar ist.

2.4 Augmentierende Pfade und Matchings

Ein *Pfad* P in G ist eine Liste von Knoten $P = v_1, v_2, \ldots, v_d$ mit $(v_i, v_{i+1}) \in E$ für $i = 1, \ldots, d-1$. Der Pfad P heißt *alternierend* (bezüglich M), wenn er abwechselnd Kanten aus M und $E \setminus M$ verwendet. Sind zusätzlich $v_1 \neq v_d$ beide frei, so ist P ein *augmentierender Pfad* (bezüglich M).

Augmentierende Pfad können verwendet werden um ein Matching zu vergrößern: Ist P ein bezüglich M augmentierender Pfad, so ist $M \Delta P$ ein Matching mit Kardinalität $|M| + 1$. Dabei bezeichnet $A \Delta B = (A \cup B) \setminus (A \cup B)$ die symmetrische Differenz von zwei Menge A und B. Abbildung 2.3 zeigt ein Beispiel für einen augmentierenden Pfad im Graphen aus Abbildung 2.1 (a).

Ein Matching M ist genau dann ein größtes Matching, wenn es keinen bezüglich M augmentierenden Pfad gibt [Ber57]. Augmentierende Pfade können effizient gefunden werden. In bipartiten Graphen benötigt das Finden eines augmentierenden Pfades $O(m)$ Zeit, in allgemeinen Graphen wird $O(m\alpha(n))$ Zeit benötigt [Tar83]. Dabei bezeichnet α die inverse Ackermannfunktion. Sie wächst so langsam, dass für alle praktischen Werte von n gilt: $\alpha(n) \leq 4$ [CLRS01].

Ein einfacher Algorithmus zum Finden eines größten Matchings besteht also darin mit einem leeren Matching zu beginnen und dieses schrittweise mit Hil-

fe von augmentierenden Pfaden zu vergrößern. Das ergibt eine Laufzeit von $O(nm\alpha(n))$ [KN05]. Der beste bekannte Algorithmus zu Berechnung größter Matchings in allgemeinen Graphen von Micali und Vazirani [MV80] basiert ebenfalls auf dieser Technik, sucht aber in einem Schritt mehrere augmentierende Pfade und benötigt daher nur $O(\sqrt{n}m)$ Zeit.

3 Matchings in Bäumen

In diesem Kapitel sei G ein Baum mit n Knoten. Da Bäume zusammenhängend und azyklisch sind, hat G genau $n-1$ Kanten. Ohne weitere Voraussetzungen lassen sich keine interessanten Aussagen über Matchings in Bäumen beweisen, da es für jedes n einen Baum mit n Knoten gibt, der ein größtes Matching der Größe 1 hat: den Stern. Abbildung 3.1 zeigt einen solchen Graphen.

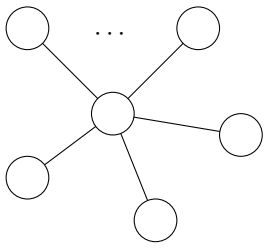


Abbildung 3.1: Ein Stern mit n Knoten hat ein größtes Matching der Größe 1

Eine sinnvolle Einschränkung für Bäume ist, den Knotengrad zu beschränken. Daher werden im Folgenden nur Bäume mit maximalem Knotengrad k betrachtet. Im Falle von $k = 3$ sind das zum Beispiel Binärbäume, dort hat jeder Knoten höchstens 3 Nachbarn: Elternknoten, linkes Kind, rechtes Kind.

Im Folgenden wird ein Algorithmus angegeben, der größte Matchings in Bäumen bestimmt. Außerdem wird eine untere Schranke für die Kardinalität von größten Matchings in Bäumen mit beschränktem Maximalgrad bewiesen.

3.1 Der Algorithmus

In Bäumen kann ein größtes Matching in Linearzeit gefunden werden: Zu Beginn wählt man einen beliebigen Knoten w als Wurzel und führt von dort aus eine Tiefensuche im Baum durch. Beim Abstieg ist nichts zu tun. Wenn eine Kante uv beim Aufstieg rückwärts durchlaufen wird, so wird sie, sofern u und v beide frei sind, zum Matching hinzugefügt. Der Pseudocode für diese Prozedur ist in Algorithmus 1 dargestellt.

Abbildung 3.2 zeigt einen Baum nach dem Besuchen des linken Teilbaums der Wurzel. Die hervorgehobenen Kanten sind im Matching enthalten, die Zahlen geben die Reihenfolge der Hinzunahme der Kanten zum Matching an. Der gestrichelte Teil wurde noch nicht besucht.

Satz 2. *Algorithmus 1 findet ein größtes Matching in einem Baum mit n Knoten in $O(n)$ Zeit.*

Algorithmus 1 : MaxMatchTree(T)

Eingabe : Baum T
Ausgabe : Größtes Matching M
Beginn
  $M \leftarrow \emptyset$
  Wähle $w \in T$
  Führe Tiefensuche von w aus durch.
  Beginn
    wenn *eine Kante u nach v zurückgegangen wird* **dann**
      wenn *u und v frei* **dann**
        $M \leftarrow M \cup \{uv\}$
  Ende
  gib M **zurück**
Ende

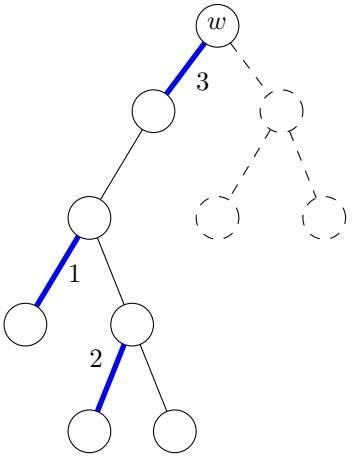


Abbildung 3.2: Situation, nachdem der linke Teilbaum der Wurzel vollständig besucht wurde.

Beweis. Der Beweis verwendet eine Charakterisierung von größten Matchings von Berge [Ber57]. Demnach ist ein Matching genau dann ein größtes Matching, wenn es keinen augmentierenden Pfad gibt.

Sei M nun die Ausgabe von Algorithmus 1 und u ein beliebiger freier Knoten in T. Ein augmentierender Pfad, der bei u beginnt verläuft entweder aufwärts oder abwärts im Suchbaum.

Fall 1: Der Pfad verläuft abwärts, also durch ein Kind v von u.

Da u frei ist, müssen nach Konstruktion alle Kinder von u, also auch v, gematcht sein. Siehe Abbildung 3.3 (a). Insbesondere kann v kein Blatt sein. Angenommen v ist mit einem Knoten s gematcht, so geht der augmentierende Pfad weiter zu s, sonst wäre er nicht alternierend. Da s aber mit seinem Elternknoten v gematcht ist, muss nach Konstruktion entweder s ein Blatt sein oder es müssen auch alle Kinder von s gematcht sein. Induktiv folgt: Der Pfad endet in einem gematchten Blatt. Daher kann der Pfad nicht augmentierend sein.

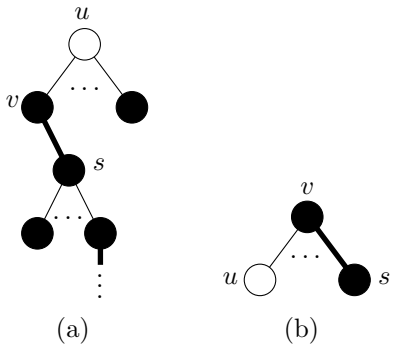


Abbildung 3.3: Möglichkeiten für den Verlauf eines augmentierenden Pfades in der Ausgabe von Algorithmus 1.

Fall 2: Der augmentierende Pfad verläuft von u zu dessen Elternknoten v.

Da u nicht zu v gematcht ist, muss nach Konstruktion v zu einem seiner anderen Kinder gematcht sein. Der augmentierende Pfad läuft also von v aus abwärts. Das ist aber nach dem ersten Fall nicht möglich. Siehe Abbildung 3.3 (b).

Es gibt also keinen augmentierenden Pfad, und das Matching ist nach Berge ein Matching maximaler Kardinalität. Da jede Kante nur zweimal betrachtet wird, einmal beim Abstieg und einmal beim Aufstieg, und es $n-1$ Kanten gibt, ist die Laufzeit in $O(n)$. □

Eine andere Möglichkeit, das Verfahren zu beschreiben ist wie folgt:

Jeder Baum mit $n > 1$ Knoten hat mindestens zwei Blätter. Man matcht stets eines der Blätter mit seinem Elternknoten und löscht das Blatt zusammen mit seinem Elternknoten aus dem Graphen. Diese Vorgehensweise wendet man nun rekursiv auf die Bäume des danach entstehenden Waldes an. Das erhaltene Matching ist ein größtes Matching.

Denn: Sei u ein Blatt in G mit Elternknoten v. Sei M' ein beliebiges größtes Matching in $G' := G - \{u, v\}$. Dann existiert nach Berge [Ber57] bezüglich M' kein augmentierender Pfad in G'. Angenommen es gäbe bezüglich $M := M' \cup \{uv\}$ einen augmentierenden Pfad in G, so kann dieser nicht durch das gematchte Blatt u verlaufen. Also müsste er innerhalb von G' verlaufen, im Widerspruch zur Maximalität von M' bezüglich G'. Demnach ist M ein größtes Matching in G.

3.2 Eine Schranke für größte Matchings in Bäumen

Dieses Verfahren liefert auch eine einfache Möglichkeit die Größe eines größten Matchings in einem Baum mit Maximalgrad k abzuschätzen.

Satz 3. *In einem Baum mit Maximalgrad k gibt es immer ein Matching M mit $|M| \geq \lceil (n-1)/k \rceil$.*

Beweis. In jedem Schritt des obigen Verfahrens wird das Matching um eine Kante vergrößert. Dadurch, dass man stets Blätter mit ihrem Elternknoten matcht und der Elternknoten höchstens Grad k hat, werden in jedem Schritt höchstens k Kanten entfernt. Das Verfahren terminiert also frühestens nach $\lceil m/k \rceil = \lceil (n-1)/k \rceil$ Schritten. □

Diese Schranke ist auch scharf, wie man leicht an einem Baum vom Grad k sieht, also einem Baum, bei dem jeder innere Knoten Grad k hat. Abbildung 3.4 zeigt einen solchen Baum für $k = 3$. Die hervorgehobenen Kanten bilden ein größtes Matching.

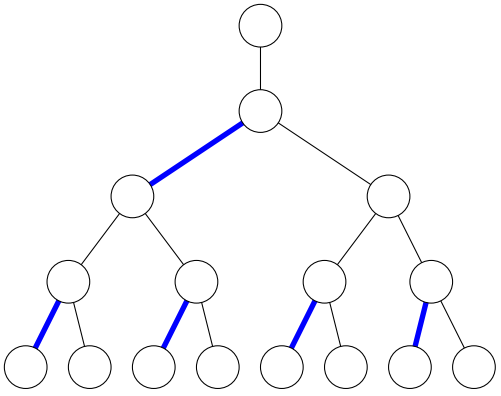


Abbildung 3.4: Baum mit Maximalgrad 3 und einem größten Matching der Kardinalität $(n-1)/3$.

Als Korollar erhält man für $k = 3$ erneut die Schranke $(n-1)/3$ aus [BDD+04]. Zusätzlich kann ein solches Matching aber nun schnell berechnet werden.

Korollar 1. *Sei G ein Graph mit Maximalgrad 3. Dann besitzt G ein Matching der Größe $(n-1)/3$. Ein Matching dieser Größe kann in $O(n)$ Zeit gefunden werden.*

Beweis. Sei T ein Spannbaum von G. Da G Maximalgrad 3 besitzt, hat auch T Maximalgrad 3. Ein solcher Spannbaum kann durch Tiefensuche in $O(n)$ Zeit bestimmt werden. Ein Matching in T ist auch ein Matching in G. In T kann ein Matching der Größe $(n-1)/3$ mit Algorithmus 1 in $O(n)$ Zeit gefunden werden. □

In 3-regulären Graphen, also Graphen, in denen jeder Knoten Grad 3 hat, existiert sogar stets ein Matching der Kardinalität $(4n-1)/9$ [BDD+04]. Diese Schranke impliziert auch eine untere Schranke für jedes nicht erweiterbare Matching, nämlich $(4n-1)/18$. Die Schranke $(n-1)/3$ aus Korollar 1 ist zwar schwächer als $(4n-1)/9$ aber immerhin besser als $(4n-1)/18$. Das angegebene Verfahren arbeitet also besser als beliebiges Hinzunehmen von erlaubten Kanten. Im nächsten Kapitel wird gezeigt, wie in 3-regulären Graphen in Linearzeit ein Matching der Größe $(3n-2)/8$ gefunden werden kann. In Kapitel 6 wird gezeigt, wie auch die Schranke $(4n-1)/9$ erreicht werden kann. Allerdings benötigt das dort angegebene Verfahren $O(n \log^4 n)$ Zeit.

4 Matchings in Graphen mit Maximalgrad k

Im Folgenden werden die Restriktionen an die betrachtete Graphenklasse gelockert und nun beliebige (zusammenhängende) Graphen mit Maximalgrad k betrachtet.

4.1 Bekannte Resultate

Eine einfache Abschätzung [BDD+04] zeigt, dass in einem Graphen $G = (V, E)$ mit n Knoten und m Kanten mit Maximalgrad k jedes nicht erweiterbare Matching mindestens $m/(2k-1)$ Kanten enthält. Dies ist natürlich zugleich eine untere Schranke für die Kardinalität eines größten Matchings. Der Beweis ist recht anschaulich und wird daher hier geführt.

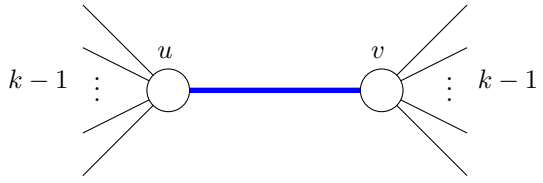


Abbildung 4.1: Eine Kante in einem Graphen mit Maximalgrad k.

Lemma 2 ([BDD+04])**.** *Ist G ein Graph mit m Kanten und Maximalgrad k, so hat jedes nicht erweiterbare Matching in G mindestens Größe $\lceil m/(2k-1) \rceil$.*

Beweis. Wähle eine beliebige Kante $e = \{u, v\} \in E$. Da u und v höchstens Grad k haben, fallen durch Entfernen der Knoten u und v aus G höchstens $2k-1$ Kanten weg (siehe Abbildung 4.1). Beliebiges Hinzufügen der Kanten zu einem Matching terminiert also frühestens nach $\lceil m/(2k-1) \rceil$ Schritten. Also hat jedes nicht erweiterbare Matching und damit auch jedes kardinalitätsmaximale Matching mindestens diese Größe. □

4.2 Eine leichte Verbesserung

Die in Abschnitt 4.1 beschriebene Schranke ist sehr pessimistisch. Immerhin sind nach einer Weile die Grade der verbleibenden Knoten kleiner als k. Bei

geschickterer Wahl der Kante, die ins Matching aufgenommen wird, kann eine bessere Abschätzung für die Kardinalität eines größten Matchings gezeigt werden.

Dazu geht man wie folgt vor: Man nimmt stets eine Kante, die an einen Knoten mit minimalem Grad angrenzt, ins Matching auf und entfernt danach die beiden angrenzenden Knoten aus dem Graphen.

Satz 4. *Sei G ein zusammenhängender Graph mit m Kanten und Maximalgrad k. Das oben angegebene Verfahren liefert dann ein Matching mit mindestens $\lceil (m-1)/(2k-2) \rceil$ Kanten.*

Beweis. Angenommen, es gibt einen Knoten v in G mit $\deg(v) \leq k-1$. Dann entfernen wir bereits im ersten Schritt höchstens $2k-2$ Kanten. Durch das Entfernen der Kanten aus dem Graphen sinkt der Grad einiger Knoten. Da jede Zusammenhangskomponente des neuen Graphen einen Knoten mit Grad höchstens $k-1$ enthält, kann der minimale Knotengrad nicht wieder auf k anwachsen. Es werden also in jedem Schritt höchstens $2k-2$ Kanten entfernt. Dies liefert eine Schranke von $m/(2k-2)$.

Gibt es hingegen keinen Knoten mit Grad kleiner als k, so nehmen wir im ersten Schritt eine beliebige Kante ins Matching auf und entfernen dadurch $2k-1$ Kanten. Dadurch wird der Grad einiger verbliebener Knoten abgesenkt. Jede der Komponenten dieses Graphen enthält nun einen Knoten mit Grad höchstens $k-1$ und wir befinden uns im ersten Fall. Das gefundene Matching hat also mindestens die Größe $(m-(2k-1))/(2k-2)+1 = (m-1)/(2k-2)$. □

Auch diese Abschätzung scheint noch sehr pessimistisch zu sein, denn nach Entfernen einiger Kanten sinkt auch der maximale Knotengrad und die Größe des gefundenen Matchings steigt an.

Für einen 3-regulären Graphen ergibt sich mit dieser Schranke ein Matching der Größe $(3n/2-1)/(2\cdot 3-2) = (3n-2)/8$. Biedl et al. [BDD+04] zeigen eine Schranke von $(4n-1)/9$, geben aber kein Verfahren an, um ein Matching dieser Größe schneller zu berechnen als durch Berechnung eines größten Matchings. Kapitel 6 zeigt ein Verfahren zum Erreichen dieser Schranke. Es läuft in leicht superlinearer Zeit.

4.3 Implementierung

Das Verfahren lässt sich wie folgt implementieren: Es werden k Listen von Knoten angelegt. Dabei wird Knoten v mit Grad i in Liste i einsortiert. Damit kann in $O(k)$ Zeit die nicht-leere Liste von Knoten mit kleinstem Grad bestimmt werden. Nun wird ein beliebiger Knoten u aus dieser Liste entnommen und eine beliebige Kante uv, die zu u inzident ist, ins Matching aufgenommen. Anschließend werden u und v aus der Liste entfernt, die maximal $2k-1$ zu u und v inzidenten Knoten aktualisiert und in die ihrem neuen Grad entsprechende Liste verschoben beziehungsweise komplett entfernt, wenn der Grad auf 0 gesunken ist. Das Hinzunehmen einer Kante benötigt also $O(k)$ Zeit.

Bei n Knoten können höchstens $n/2$ Kanten in das Matching aufgenommen werden. Daher benötigt diese Implementierung $O(kn)$ Zeit. Nimmt man den Maximalgrad k als kleine Konstante an, so ist die Laufzeit $O(n)$.

4.4 Verbesserung in planaren Graphen

Im Falle eines planaren Graphen lässt sich die Schranke noch weiter verbessern:

Satz 5. *In einem planaren Graphen $G = (V, E)$ mit $|E| = m$ und Maximalgrad k gibt es stets ein Matching der Mindesgröße $\lceil m/(k+4) \rceil$. Ein solches Matching kann in $O(n)$ Zeit berechnet werden.*

Beweis. Die Abschätzung verläuft im Wesentlichen wie zuvor. Allerdings wird die Tatsache ausgenutzt, dass in einem planaren Graphen stets ein Knoten v mit $\deg(v) \leq 5$ existiert [KN05]. Fügt man eine zu einem solchen Knoten v inzidente Kante uv zum Matching hinzu und entfernt u und v aus dem Graphen, so ist der resultierende Graph wieder planar und es wurden höchstens $k+5-1 = k+4$ Kanten entfernt. Das Verfahren liefert also ein inklusionsmaximales Matching mit mindestens $\lceil m/(k+4) \rceil$ Kanten. □

Diese Schranke ist für $k > 5$ besser als die zuvor angegebenen Schranken. Ein Matching dieser Größe kann mit dem oben angegebenen Algorithmus für beliebige Graphen mit Maximalgrad k bestimmt werden.

4.5 Ein anderer Ansatz

Ein anderer Ansatz zur Berechnung von Matchings in Graphen mit beschränktem Maximalgrad ergibt sich, wenn man versucht bereits bekannte Techniken aus Kapitel 3 zu verwenden. Die Grundidee hierbei ist, zunächst einen spannenden Baum in einem Graphen zu suchen und dann den Matching-Algorithmus für Bäume zu verwenden.

In diesem Fall könnte eine Anwendung wie folgt aussehen: Bestimme einen beliebigen Spannbaum T des Graphen G. Da G Maximalgrad k hat, hat auch T Maximalgrad k. Das Verfahren aus Abschnitt 3.1 findet also ein Matching der Kardinalität mindestens $(n-1)/k$.

Leider ist diese Schranke im Vergleich zu $(m-1)/(2k-2)$ relativ schwach. Zudem ist es auch nicht möglich, die Abschätzung für dieses Verfahren auf der gegebenen Graphenklasse zu verbessern, da der in Abschnitt 3.2 beschriebene Worst-Case gerade ein Graph mit Maximalgrad k ist. Da Bäume inbesondere planare Graphen sind, hilft es auch nicht die Schranke zu verbessern, wenn man zusätzlich noch Planarität fordert.

Die Abschätzung wird schlecht, weil der Maximalgrad im Spannbaum recht groß ist. Gelänge es einen Spannbaum mit kleinem Maximalgrad zu konstruieren, so ergäbe sich auch eine bessere Schranke. Leider kann die Existenz eines

„guten“ Spannbaums nicht immer garantiert werden. Zudem ist es schwer einen Spannbaums mit kleinstem Maximalgrad in beliebigen Graphen zu finden: Ein Spannbaum mit Maximalgrad 2 ist ein Hamilton-Pfad. Sogar in 3-regulären, planaren und dreifach zusammenhängenden Graphen ist es NP-schwer zu entscheiden, ob ein Hamiltonpfad/kreis existiert [GJT76]. Es ist auch NP-schwer zu entscheiden, ob ein Graph einen zusammenhängenden spannenden Teilgraphen mit Maximalgrad $r, r \geq 2$ besitzt [Yan81]. Es gibt aber Approximationsalgorithmen, die einen Spannbaum finden, dessen Maximalgrad höchstens um 1 größer ist, als der Maximalgrad eines optimalen Spannbaums [FR92]. Die Laufzeit dieses Algorithmus ist allerdings zu hoch, um für die Berechnung von Matchings interessant zu sein. Um die Idee weiter zu verfolgen, benötigt man also zusätzliche Voraussetzungen, um die Existenz von Spannbäumen mit kleinem Maximalgrad zu sichern, sowie effiziente Verfahren um solche Spannbäume zu konstruieren.

5 Mehrfach zusammenhängende Graphen mit Maximalgrad k

Czumaj und Strothmann [CS97] zeigten, dass in einem zweifach zusammenhängenden Graphen G mit Maximalgrad k stets ein Spannbaum T mit Maximalgrad $\Delta T \leq \lceil (k+2)/2 \rceil$ existiert. Ein solcher Baum kann deterministisch in $O(m + n \log^4 n)$ Zeit gefunden werden, randomisiert ist sogar eine Laufzeit von $O(m + n \log^2 n)$ möglich. Ist G zusätzlich planar, so lässt sich die Laufzeit auf $O(n \log n)$ verbessern. Im folgenden Abschnitt wird dieses Verfahren vorgestellt. Anschließend wird das Resultat zum Finden von Matchings verwendet.

5.1 Konstruktion von Spannbäumen mit kleinem Maximalgrad

Zur Konstruktion werden sogenannte *wohlstrukturierte Graphen* verwendet, da sich die Darstellung so einfacher gestaltet. Es ist aber leicht aus einem beliebigen zweifach zusammenhängenden Graphen einen wohlstrukturierten Graphen zu konstruieren.

Definition 5.1. *Ein* wohlstrukturierter Graph (ws-Graph) *ist ein bipartiter zusammenhängender Graph $G(V_1, V_2, E)$ zusammen mit einer Beschriftungsfunktion L und einigen ausgezeichneten Knoten s, t, u, x. Zusätzlich sollen folgende Eigenschaften gelten:* $\deg_G(v_1) = 2$ *für alle $v_1 \in V_1$ und nach Hinzufügen der Kante (s, x) zu G soll der Graph zweifach zusammenhängend sein.*

Die Beschriftungsfunktion $L : V_2 \longrightarrow \{\boxed{0}, \boxed{1}, \boxed{2}\}$ ordnet jedem Knoten in V_2 ein Label zu. Auf den Labels wird durch $\boxed{0} \prec \boxed{1} \prec \boxed{2}$ eine Totalordnung definiert. Zusätzlich soll die Beschriftung folgende Eigenschaften erfüllen:

- *Gibt es einen Knoten mit Label $\boxed{0}$, so ist er der einzige $\boxed{0}$-Knoten und es gibt höchstens einen anderen Knoten mit Label $\boxed{1}$.*
- *Gibt es keinen Knoten mit Label $\boxed{0}$, so gibt es höchstens 3 Knoten mit Label $\boxed{1}$.*
- *Gibt es einen Knoten mit Label $\boxed{0}$, so hat er geraden Grad.*
- *Alle Knoten mit Label $\boxed{1}$ haben ungeraden Grad.*

Für die ausgezeichneten Knoten $s, t, u \in V_2$ und $x \in \{t, u\}$ gelten folgende Eigenschaften:

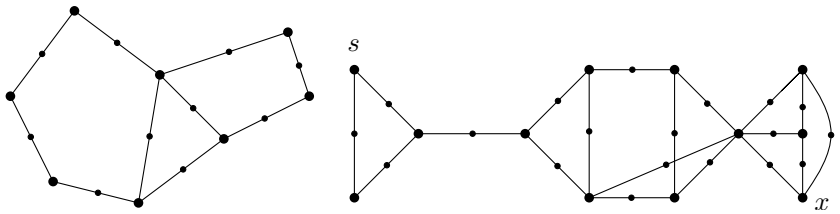


Abbildung 5.1: Zweifach zusammenhängender ws-Graph (a) und nicht zweifach zusammenhängender ws-Graph (b). Kleine Knoten sind in V_1, große Knoten in V_2.

- *Ist* $|V_2| \geq 3$*, so sind* s, t, u *paarweise verschieden und* s *ist ein Knoten mit dem kleinsten Label,* t *ein Knoten mit dem zweitkleinsten Label und* u *ein Knoten mit dem drittkleinsten Label in* G*.*
- x, s *sind keine Separatorknoten, nach Entfernen eines dieser Knoten zerfällt der Graph also nicht in mehrere Komponenten.*
- *Ist* G *nicht zweifach zusammenhängend, so befinden sich* s *und* x *in verschiedenen Blättern des* 2*-Blockbaums* $\mathbb{BCT}(G)$*.*

Für einen bipartiten Graphen $G = (V_1(G), V_2(G), E(G))$ wird die Gesamtmenge seiner Knoten $V_1(G) \cup V_2(G)$ mit $V(G)$ bezeichnet. Abbildung 5.1 zeigt zwei Beispiele für ws-Graphen.

Definition 5.2. *Ein* ws-Spannbaum T *eines ws-Graphen* G *ist ein spannender Baum von* G*, so dass jeder Knoten* v *mit Label* $\boxed{i}$ *in* T *höchstens Grad* $\lceil (\deg_G(v) + i)/2 \rceil$ *besitzt. Alle ungelabelten Knoten können in* T *beliebigen Grad haben.*

In einem ws-Graph kann per Definition durch Hinzufügen einer einzigen Kante Zweifachzusammenhang hergestellt werden. Der folgende Satz sagt etwas über die Struktur solcher Graphen aus:

Satz 6 ([Tut84])**.** *Sei* G *ein zweifach zusammenhängender Graph mit mindestens zwei Kanten und sei* e *eine Kante von* G*. Dann ist entweder* $G - e$ *zweifach zusammenhängend oder der* 2*-Blockbaum von* G *ist ein Pfad und* e *ist zu jeweils einem Knoten in den Blättern des* 2*-Blockbaums inzident.*

Wenn ein ws-Graph also nicht zweifach zusammenhängend ist, so ist sein 2-Blockbaum $\mathbb{BCT}(G)$ ein Pfad. Zudem ergibt sich induktiv folgendes Korollar:

Korollar 2 ([Str97])**.** *Sei* G *ein zweifach zusammenhängender ws-Graph und* s *der Knoten mit dem kleinsten Label. Dann hat der* 2*-Blockbaum von* $G - s$ *genau* $deg_G(s)$ *Blätter.*

Der Algorithmus zur Bestimmung eines Spannbaums mit kleinem Maximalgrad arbeitet rekursiv und verwendet eine Reihe von Zerlegungsoperationen, die den Graphen entweder verkleinern oder in zwei Komponenten zerlegen, die selbst wieder wohlstrukturierte Graphen sind. Durch eine geschickte Wahl der

Beschriftung in den Komponenten können die ws-Spannbäume der Teilgraphen dann zu einem ws-Spannbaum des gesamten Graphen zusammengesetzt werden. Im nächsten Abschnitt werden drei grundlegende Zerlegungsoperationen definiert.

5.2 Zerlegungsoperationen

Die erste Zerlegungsoperation teilt die inzidenten Kanten eines Separatorknotens mit Grad 2 im 2-Blockbaum so auf, dass die Kanten der beiden inzidenten Blöcke in verschiedenen Graphen sind. Die zweite Zerlegungsoperation heißt DELETE_AND_COMBINE und ersetzt in einem zweifach zusammenhängenden Graphen einen Pfad $v_2 \to x_2 \to s \to x_1 \to v_1$ durch einen Pfad $v_2 \to z \to v_1$. Dabei haben die Knoten x_1, x_2 Grad 2 und z ist ein neuer Knoten. Die letzte Operation verwendet ein separierendes Paar zur Zerlegung. Der Graph wird dadurch in zwei Komponenten zerlegt, die beide zweifach zusammenhängend sind.

In allen drei Fällen wird gezeigt, wie die Beschriftung gewählt werden muss, um einen ws-Graphen zu erhalten und wie ein ws-Spannbaum von G aus einem ws-Spannbaum des zerlegten Graphen konstruiert werden kann.

Lemma 3. *Sei G ein ws-Graph und $a \in V_2$ ein Separatorknoten mit $L(a) \in \{\boxed{1}, \boxed{2}\}$, der zu zwei Zweifachzusammenhangskomponenten inzident ist. Seien G_1 und G_2 die Graphen, die durch Zerlegung an a entstehen. Jeder der nachfolgenden Fälle weist a in G_1 und G_2 Labels zu. Sei T_i ein ws-Spannbaum von G_i für $i = 1, 2$. Dann ist die Vereinigung der Spannbäume T_1, T_2 ein ws-Spannbaum von G.*

1. *Ist $L(a) = \boxed{2}$ und sind $\deg_{G_1}(a)$ und $\deg_{G_2}(a)$ gerade, so erhält a in einer der beiden Komponenten das Label $\boxed{2}$ und in der anderen das Label $\boxed{0}$.*
2. *Ist $L(a) = \boxed{2}$ und sind $\deg_{G_1}(a)$ und $\deg_{G_2}(a)$ ungerade, so erhält a in beiden Graphen das Label $\boxed{1}$.*
3. *Ist $L(a) = \boxed{2}$ und (ohne Beschränkung der Allgemeinheit) $\deg_{G_1}(a)$ gerade und $\deg_{G_2}(a)$ ungerade, so erhält a entweder in G_1 das Label $\boxed{2}$ und in G_2 $\boxed{1}$ oder in G_1 $\boxed{0}$ und in G_2 $\boxed{2}$.*
4. *Ist $L(a) = \boxed{1}$, dann (ohne Beschränkung der Allgemeinheit) $\deg_{G_1}(a)$ gerade und $\deg_{G_2}(a)$ ungerade und a erhält in G_1 das Label $\boxed{0}$ und in G_2 $\boxed{1}$.*

Beweis. Es ist klar, dass die Graphen G_i durch die angegebenen Beschriftungen zu ws-Graphen werden und dass die Vereinigung der ws-Spannbäume T_i einen Spannbaum von G ergibt. Es muss nur der Grad von a in $T := T_1 \cup T_2$ überprüft werden, um sicherzustellen, dass es sich bei T um einen ws-Spannbaum von G handelt.

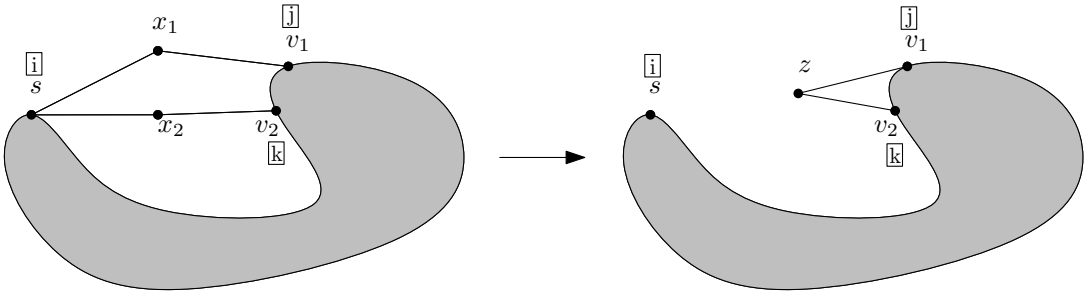


Abbildung 5.2: DELETE-AND-COMBINE x_1, x_2 bei s.

In Fall 1 sei etwa $\deg_{G_1}(a) = 2i$ und $\deg_{G_2}(a) = 2j$ beide gerade. Dann gilt mit der oben angegebenen Beschriftung: $\deg_{T_1}(a) \leq i + 1$ und $\deg_{T_2}(a) \leq j$. Also gilt für den Grad von a:

$$\deg_T(a) \leq i + j + 1 = \frac{(2i + 2j) + 2}{2} = \frac{\deg_G(a) + 2}{2}.$$

Alle anderen Fälle lassen sich analog beweisen. □

Das nächste Lemma zeigt die zweite Zerlegungsoperation:

Lemma 4. *Sei $G = (V_1, V_2, E, L, s, t, u, x)$ ein ws-Graph mit $|V_2| > 1$. Seien x_1, x_2 zwei zu s adjazente Knoten. Seien v_1, v_2 der jeweils von s verschiedene andere zu x_1, x_2 adjazente Knoten. Sei G' der Graph, der aus G entsteht, indem zunächst x_1, x_2 entfernt werden. Anschließend wird ein neuer Knoten z zusammen mit Kanten zv_1 und zv_2 hinzugefügt. Die Beschriftung bleibt unverändert. Der Ergebnisgraph wird mit $G\langle x_1, x_2, s\rangle$ bezeichnet. Die Zerlegungsoperation heißt DELETE_AND_COMBINE, Abbildung 5.2 zeigt ein Beispiel.*

Aus einem ws-Spannbaum T' von G' kann ein ws-Spannbaum von G konstruiert werden. Ist G zweifach zusammenhängend, so ist $G' + sz$ zweifach zusammenhängend.

Beweis. Um aus T' einen ws-Spannbaum von G zu konstruieren werden die Kanten sx_1 und sx_2 zu T' hinzugefügt. Ist außerdem die Kante $v_iz, i = 1, 2$ in T', so wird x_iv_i zu T' hinzugefügt. Anschließend wird z mit allen inzidenten Kanten entfernt. Der resultierende Graph hat genau einen Kreis, der a und s enthält. Es liegt also eine der neuen Kanten sx_1, sx_2 auf dem Kreis. Entfernen der entsprechenden Kante erhält die Gradbeschränkungen und ergibt einen ws-Spannbaum T von G. □

Lemma 5. *Sei G ein zweifach zusammenhängender ws-Graph, $\deg_G(s) \geq 4$ und seien x_1, x_2, x_3 drei zu s adjazente Knoten. Setze $G^* := G - \{sx_1, sx_2, sx_3\}$. Angenommen G^* besitzt genau einen Separatorknoten a mit $\deg_{\mathbb{BCT}(\mathbb{G}^*)}(a) = 4$.*

Sei z ein neuer Knoten. Wird $G - \{sx_1, sx_2\} \cup \{zx_1, zx_2\}$ an a in zwei Graphen G_1, G_2 zerlegt, mit $a \in G_1, z \in G_2$, so können Labels zu s und a in G_1 und z und a in G_2 zugewiesen werden, so dass folgende Eigenschaften gelten:

- *G_1, G_2 sind ws-Graphen.*

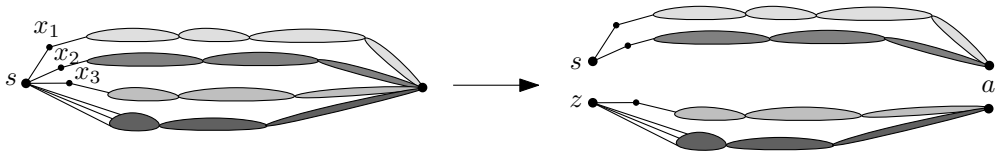


Abbildung 5.3: Zerlegung eines separierenden Paares.

- *Aus ws-Spannbäumen T_1, T_2 von G_1, G_2 kann ein ws-Spannbaum von G konstruiert werden.*

Eine solche Zerlegung ist in Abbildung 5.3 dargestellt. Da die Beschriftung sowohl vom Label von s, als auch von der Lage der anderen Knoten mit kleinsten Labels, t, u abhängt, gibt es viele Fälle zu beachten. Auf eine genaue Darstellung wird daher verzichtet.

Der Beweis ist ebenfalls bei Strothmann zu finden [Str97].

5.3 Der Algorithmus

Der Algorithmus unterscheidet zunächst, ob G zweifach zusammenhängend ist oder nicht. Ist dies nicht der Fall, so wird ein Separatorknoten verwendet um den Graphen geeignet zu zerlegen. Andernfalls wird der Knoten mit kleinstem Label s verwendet um den Graphen zu zerlegen oder zu verkleinern. Als nächstes folgt eine grobe Übersicht über den Ablauf des Algorithmus, die genauen Zerlegungsoperationen werden anschließend erklärt.

Algorithmus ws-SPANNBAUM
Eingabe: ws-Graph $G = (V_1(G), V_2(G), E(G), L_G, s_G, t_G, u_G, x_G)$ und ein Wald $F = (V(F), E(F))$.
Ausgabe: Modifizierter Wald F, so dass F eingeschränkt auf $V(G)$ ein ws-Spannbaum von G ist.

(1) Wenn s ein $\boxed{2}$-Knoten ist und $\deg_G(s)$ gerade ist, gib s das Label $\boxed{0}$, andernfalls $\boxed{1}$.
(2) Wenn $|V_2(G)| \leq 2$ dann (a); return (F);
(3) $a :=$ Separatorknoten, der s und x in G separiert.
(4) Existiert ein solcher Knoten a, dann (b); return (F);
(5) Wenn $\deg_{\mathbb{BCT}}(s) = 2$, dann (c); return (F);
(6) Seien $sx_i, i = 1, 2, 3$ drei zu s inzidente Kanten und x_iv_i die von sx_i verschiedene zu x_i. inzidente Kante. Sei $K := \{v_1, v_2, v_3\}$ und $X := \{x_1, x_2, x_3\}$.
(7) Ist $\deg_G(s) = 3$, dann (d); return (F);
(8) Wenn $|K| = 1$, dann (e); return (F);
(9) Wenn $|K| = 2$, dann (f); return (F);
(10) (g); return(F);

(a) Das ist der Basisfall. Sei $V_2 = \{s, v\}$, $V_1 = \{x_1, ..., x_d\}$ und $d := \deg_G(s)$. Ist d gerade, so hat s Label $\boxed{0}$ und v muss Label $\boxed{2}$ besitzen, da es nur einen Knoten mit Label $\boxed{0}$ geben darf und Knoten mit Label $\boxed{1}$ ungeraden

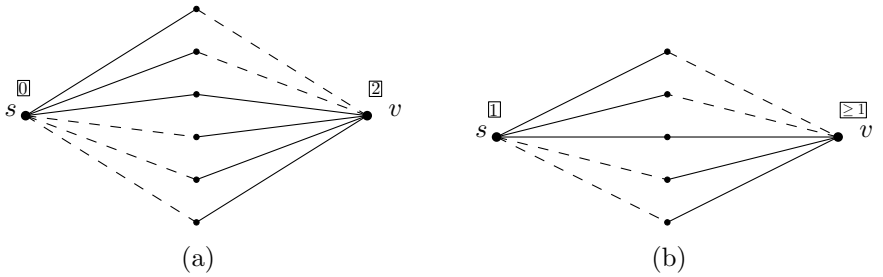


Abbildung 5.4: Basisfall für ws-Spannbäume in ws-Graphen

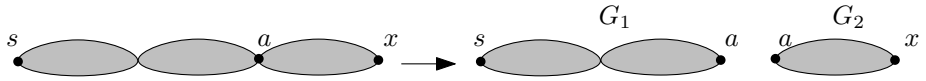


Abbildung 5.5: Zerlegung eines ws-Graphen mittels eines Separatorknotens.

Grad besitzen müssen. Daher induzieren die Kanten $sx_1, \ldots, sx_{d/2}$ und $vx_{d/2}, \ldots, vx_d$ einen ws-Spannbaum von G. Die Situation ist in Abbildung 5.4 (a) dargestellt.

Ist d ungerade, so induzieren die Kanten $sx_1, \ldots, sx_{(d+1)/2}$ zusammen mit den Kanten $vx_{(d+1)/2}, \ldots, vx_d$ einen ws-Spannbaum von G, da v mindestens Label $\boxed{1}$ haben muss. Siehe Abbildung 5.4 (b).

(b) $|V_2| > 2$, G ist nicht zweifach zusammenhängend und es gibt einen Separatorknoten a, der s von x separiert. Da $G+sx$ zweifach zusammenhängend ist, muss der 2-Blockbaum von G ein Pfad sein und s und x müssen in verschiedenen Blättern des 2-Blockbaums sein. Man zerlegt G bei a in zwei Komponenten G_1, G_2, dabei sei $s \in V(G_1)$. Dann sind $G_1 + sa$ und $G_2 + ax$ zweifach zusammenhängend. Siehe Abbildung 5.5.

Je nach Beschriftung der einzelnen Knoten muss die Beschriftung der neuen Komponenten anders aussehen.

(i) Ist $L_G(s) = \boxed{0}$, so gilt $L_G(a) = \boxed{2}$ und a erhält in G_1 mindestens Label $\boxed{1}$ und in G_2 mindestens Label $\boxed{0}$.

(ii) Ist $L_G(s) = \boxed{1}$ und $L_G(a) = \boxed{1}$, so erhält a in G_1 und G_2 mindestens Label $\boxed{0}$.

(iii) Andernfalls gilt $L_G(a) = \boxed{2}$ und $L_G(s) = \boxed{1}$. In diesem Fall gibt es keinen Knoten mit Label $\boxed{0}$. Dann darf es in einer Komponente, die einen Knoten mit Label $\boxed{0}$ enthält, höchstens einen Knoten mit Label $\boxed{1}$ geben. Da s, t, u die drei Knoten mit den kleinsten Labels sind, ist dieser Fall abhängig von der Lage von t und u. Sei $y := \{t, u\} \setminus \{x\}$. Gilt $y \subseteq V(G_i)$, so erhält a in G_j mindestens Label $\boxed{1}$ und in der anderen Komponente mindestens Label $\boxed{0}$.

Nach Lemma 3 sind G_1, G_2 ws-Graphen und die Vereinigung ihrer ws-Spannbäume ein ws-Spannbaum von G.

(c) Es gilt $|V_2| > 2$, G ist zweifach zusammenhängend und $\deg_G(s) = 2$. Seien x_1, x_2 die beiden Nachbarn von s. Sei v_i der von s verschiedene zu x_i adjazente Knoten. Nun werden s, x_1, x_2 mit ihren inzidenten Kanten aus

Abbildung 5.6: Entfernen von s im Fall von $\deg_G(s) = 2$.

G entfernt. Der Blockbaum des erhaltenen Graphen G_1 ist dann ein Pfad. Nach Hinzufügen der Kante v_1v_2 wäre G_1 zweifach zusammenhängend. Siehe Abbildung 5.6.

Da der $\boxed{0}$-Knoten entfernt wurde, gibt es höchstens einen $\boxed{1}$-Knoten in G. Ist einer der beiden Knoten v_1, v_2, etwa v_1 ein $\boxed{1}$-Knoten, so erhält v_1 das Label $\boxed{0}$. Hat v_2 zudem ungeraden Grad in G_1, so erhält v_2 das Label $\boxed{1}$ in G.

Sei T^* ein ws-Spannbaum von G_1. Da s Label $\boxed{0}$ hatte und das Label von v_1 reduziert wurde, kann in $T^* \cup s$ an jedem dieser Knoten eine Kante angehängt werden. Hatte v_2 geraden Grad, $\deg_G(v_2) = 2i$, so muss garantiert werden, dass $\deg_T(v_2) \leq \lceil (2i+2)/2 \rceil = i+1$. Da die Beschriftung von v_2 geändert wurde, gilt $\deg_{T^*}(v_2) \leq \lceil ((2i-1)+1)/2 \rceil = i$. Also kann in $T^* \cup s$ eine weitere Kante bei v_2 angehängt werden. Ist $\deg_G(v_2) = 2i-1$ ungerade, so muss $\deg_T(v_2) \leq \lceil ((2i-1)+2)/2 \rceil = i+1$ garantiert werden. Es gilt $\deg_{T^*} \leq \lceil ((2i-2)+2)/2 \rceil = i$ und es kann eine Kante an v_2 in $T^* \cup s$ angehängt werden.

Sind weder v_1 noch v_2 $\boxed{1}$-Knoten, so erhalten v_1, v_2 das Label $\boxed{1}$, wenn sie nun ungeraden Grad haben. G_1 enthält dann keinen $\boxed{0}$-Knoten und höchstens 3 $\boxed{1}$-Knoten. G_1 ist also ein ws-Graph. Ist T^* ein ws-Spannbaum von G_1, so ist es wie zuvor möglich an s, v_1, v_2 je eine Kante hinzuzufügen.

Daher ist $T := T^* \cup \{sx_1, x_1v_1, x_2v_2\}$ ein ws-Spannbaum von G.

(d) Es gilt $|V_2| > 2$, G ist zweifach zusammenhängend und $\deg_G(s) = 3$. Die Knoten x_1, x_2, x_3 waren die drei zu s adjazenten Knoten und v_i für $i = 1, 2, 3$ der von s verschiedene zu x_i adjazente Knoten. Die Knotenmenge K ist definiert durch $K = \{v_1, v_2, v_3\}$. Dieser Fall unterscheidet sich nach $|K|$. Zunächst gilt $|K| > 1$, da sonst entweder $|V_2| = 2$ oder G nicht zweifach zusammenhängend.

Ist $|K| = 2$, so seien v_1, v_2 zwei verschiedene Knoten von K und es sei $v_1 = v_3$. Dann ist $\{v_1, v_2\}$ ein Separator, der $\{s, x_1, x_2, x_3\}$ vom Rest von G trennt. Nun kann die zweite Zerlegungsoperation, DELETE_AND_COMBINE, mit den Knoten s, x_1, x_2 angewendet werden. Dadurch entsteht ein neuer Graph G_1, dessen 2-Blockbaum zwei triviale Blöcke $\{s, x_3\}$ und $\{x_3, v_3\}$ und einen nichttrivialen Block, der alle restlichen Kanten von G_1 enthält, besitzt. G_1 ist ein ws-Graph. Siehe Abbildung 5.7.

Ist $|K| = 3$, so besitzt der 2-Blockbaum nach Korollar 2 genau $\deg_G(s)$ Blätter. Daher existiert ein Separatorknoten a oder ein Block B mit Grad 3 im 2-Blockbaum von $G - s$. Betrachte nun die Pfade, die von a bzw. B zu den drei Blättern des 2-Blockbaums, die die Knoten x_i enthalten, führen. Da es höchstens zwei von s verschiedene $\boxed{1}$-Knoten gibt,

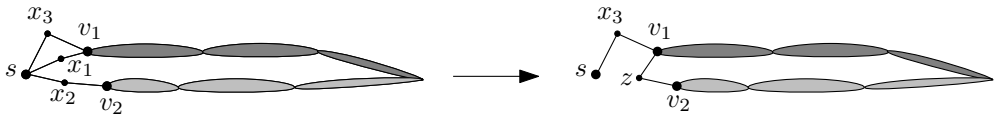


Abbildung 5.7: Zerlegung von ws-Graphen im Fall von $\deg(s) = 3, |K| = 2$.

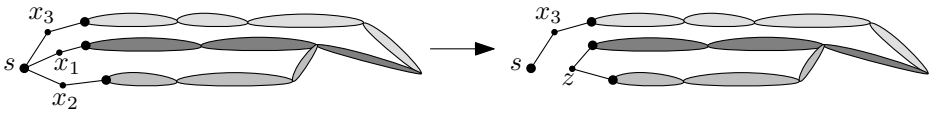


Abbildung 5.8: Zerlegung von ws-Graphen im Fall von $\deg(s) = 3, |K| = 3$.

liegt auf einem der Pfade, etwa P_3, kein $\boxed{1}$-Knoten außer a. Also liegen im Graphen $G - sx_3$ alle $\boxed{1}$-Knoten im selben Block. Sei G_1 nun der Graph, der bei Zerlegung durch DELETE_AND_COMBINE entlang der Knoten s, x_1, x_2 entsteht. Dann ist G_1 ist ein ws-Graph. Siehe Abbildung 5.8.

Die Konstruktion eines ws-Spannbaums von G aus einem ws-Spannbaum von G_1 funktioniert wie in Lemma 4.

(e) $|V_2| > 2$, G ist zweifach zusammenhängend, $\deg_G(s) \geq 4$, $|K| = 1$. Sei $K = \{v\}$. Entferne die Knoten x_1, x_2 zusammen mit ihren inzidenten Kanten. Das Vorgehen ist in Abbildung 5.9 dargestellt.

Der resultierende Graph G^* ist zweifach zusammenhängend und daher ein ws-Graph. Jeder ws-Spannbaum von T^* von G^* kann zu einem ws-Spannbaum T von G erweitert werden, indem Kanten sx_1, vx_2 hinzugefügt werden. Denn $\deg_{G^*}(s) = \deg_G(s) - 2$, $\deg_{G^*}(v) = \deg_G(v) - 2$ und die Beschriftung bleibt in G^* unverändert. Daher kann in T^* an jedem der Knoten s, v eine Kante hinzugefügt werden.

(f) $|V_2| > 2$, G ist zweifach zusammenhängend, $\deg_G(s) \geq 4$, $|K| = 2$. Sei $K = \{v_1, v_3\}$ und v_1 sei adjazent zu x_1, x_2. Führe die Reduktionsoperation DELETE_AND_COMBINE für s, x_2, x_3 durch, wie in Abbildung 5.10.

Der resultierende Graph $G\langle s, x_2, x_3\rangle$ ist zweifach zusammenhängend und daher ein ws-Graph. Die Konstruktion eines ws-Spannbaums von G aus einem ws-Spannbaum von $G\langle s, x_2, x_3\rangle$ funktioniert dann wie in Lemma 4.

(g) $|V_2| > 2$, G ist zweifach zusammenhängend, $\deg_G(s) \geq 4$, $|K| = 3$. Sei H der Graph, der durch Entfernen der Kanten $sx_i, i = 1, 2, 3$ aus G entsteht. Nach Korollar 2 hat der 2-Blockbaum von H vier Blätter. Daher gibt es entweder einen Separatorknoten oder Block mit Grad 4 oder zwei Separatorknoten/Blöcke mit Grad 3 im 2-Blockbaum von G.

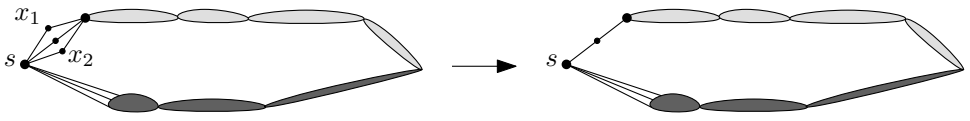


Abbildung 5.9: Zerlegung eines ws-Graphen mit $\deg_G(s) \geq 4, |K| = 1$.

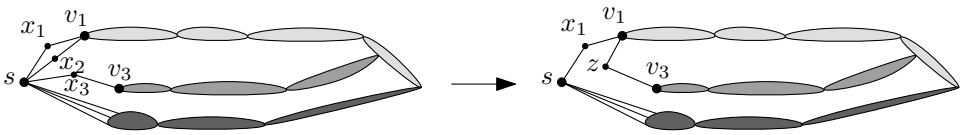


Abbildung 5.10: Zerlegung eines ws-Graphen mit $\deg_G(s) \geq 4, |K| = 2$.

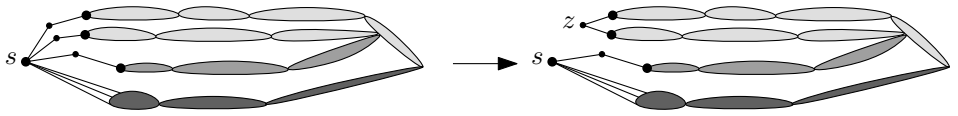


Abbildung 5.11: Zerlegung eines ws-Graphen mit $\deg_G(s) \geq 4, |K| = 3$.

Gibt es keinen Separatorknoten mit Grad 4 im 2-Blockbaum, so gibt es zwei Knoten (Separatorknoten oder Blöcke) n_1, n_2, die im 2-Blockbaum Grad 3 haben, oder einen Block mit Grad 4. Im ersten Fall kann ohne Einschränkung angenommen werden, dass n_1 auf jedem Pfad von x_α zu n_2 und auf jedem Pfad von x_β zu n_2 liegt. Dann ist nur das Ergebnis der Zerlegung entlang s, x_α, x_β nicht zweifach zusammenhängend. Also sind zwei der Zerlegungen entlang s, x_1, x_2, s, x_1, x_3 und s, x_2, x_3 zweifach zusammenhängend. Im zweiten Fall sind die Ergebnisse aller drei Zerlegungen zweifach zusammenhängend. Siehe Abbildung 5.11.

Daher ist mindestens eine der Zerlegungen entlang s, x_1, x_2 und s, x_2, x_3 zweifach zusammenhängend, oder H besitzt einen Separatorknoten mit Grad 4 im 2-Blockbaum.

Im ersten Fall kann mit Hilfe von Lemma 4 ein ws-Spannbaum von G bestimmt werden. Im zweiten Fall kann das separierende Paar $\{s, a\}$ wie in Lemma 3 aufgetrennt und ein ws-Spannbaum von G konstruiert werden.

5.4 Implementierung

Um effizient testen zu können, ob ein Graph zweifach zusammenhängend ist und gegebenfalls einen Separatorknoten zu bestimmen, der zwei gegebene Knoten separiert, wird eine spezielle Datenstruktur verwendet, die dynamisch die Zweifachzusammenhangskomponenten eines Graphen verwalten kann. Der Zeitaufwand, um zu testen, ob der Graph zweifach zusammenhängend ist und gegebenfalls einen Separatorknoten zu bestimmen, sei T_{QUERY}. Der Aufwand, um eine Kante aus dem Graphen zu entfernen, sei T_{UPDATE}.

Für die Laufzeit des Verfahrens gilt folgendes Lemma:

Lemma 6 ([CS97])**.** *Jeder ws-Graph G besitzt einen ws-Spannbaum. Ein ws-Spannbaum von G kann in $O(n \cdot T_{\text{UPDATE}} + n \cdot \log n \cdot T_{\text{QUERY}})$ Zeit gefunden werden.*

Mit der Datenstruktur aus [HdLT01] ist $T_{\text{QUERY}} = T_{\text{UPDATE}} = O(\log^4 n)$. Die Autoren geben sogar an, es sei $T_{\text{UPDATE}} = O(\log^4 n)$ und $T_{\text{QUERY}} = O(\log n)$ möglich. Damit benötigt das Auffinden eines ws-Spannbaums $O(n \log^5 n)$ beziehungsweise $O(n \log^4 n)$ Zeit.

5.5 Zweifacher Zusammenhang

Sei $G = (V, E)$ ein zweifach zusammenhängender Graph. Aus G kann ein ws-Graph konstruiert werden, indem jede Kante durch einen Knoten unterteilt wird. Die ursprünglich vorhandenen Knoten bilden dann die Menge V_2, die Knoten, die die Kanten unterteilen sind in V_1. Dadurch entsteht ein ws-Graph $G' = (V_1, V_2, E')$ mit $V_2 = V, V_1 = \{v_e : e \in E\}$ und $E' = \bigcup_{e=uw \in E} \{uv_e, v_e w\}$. Alle Knoten aus V_2 erhalten das Label $\boxed{2}$. Eine Kante von $e = uv \in E$ wird genau dann in den Spannbaum von G aufgenommen, wenn beide Kanten $uv_e, v_e w$, in die e in G' zerlegt wurde, im ws-Spannbaum von G' enthalten sind. Da jeder Knoten $v \in V$ in G den gleichen Grad wie in G' hat, entsteht dadurch ein Spannbaum T, für den gilt: $\deg_T(v) \leq \lceil (\deg_G(v)/2 \rceil + 1$.

Insgesamt ergibt sich folgender Satz:

Satz 7. *Jeder zweifach zusammenhängende Graph $G = (V, E)$ besitzt einen Spannbaum T, so dass $\deg_T(v) \leq \lceil (\deg_G(v)/2 \rceil + 1$. Ein solcher Spannbaum kann in $O(n \log^4 n)$ Zeit gefunden werden.*

Hat G Maximalgrad k, so ergibt sich damit ein Spannbaum mit Maximalgrad $\lceil k/2 \rceil + 1$.

Bemerkung 1. *Ist G zusätzlich planar, so gibt es eine Datenstruktur mit $T_{\text{UPDATE}} = O(\log^2 n)$ und $T_{\text{QUERY}} = O(\log n)$ [Rau94]. Dadurch ergibt sich im planaren Fall eine Laufzeit von $O(n \log^2 n)$ bei linearem Speicheraufwand. Mit quadratischem Speicheraufwand ist sogar eine Laufzeit von $O(n \log n)$ möglich [Str97].*

Zum Erreichen der Laufzeit $O(n \log n)$ wird natürlich nicht wirklich eine quadratische Speichermenge benötigt. Sie wird angefordert, aber nicht initialisiert. Zusätzlich wird ein Feld linearer Größe mit Zeigern auf alle bereits initalisierten Einträge mitgeführt.

Für jede Speicherzelle wird neben den Nutzdaten auch ein Zeiger in die Liste der initalisierten Zellen mitgeführt. Wird auf eine Speicherzelle zugegriffen, so kann nun in konstanter Zeit entschieden werden, ob sie bereits initialisiert ist: Zunächst wird der zugehörige Zeiger verfolgt. Zeigt er nicht in das Feld der initialisierten Speicherzellen, so ist die Zelle nicht initalisiert. Zeigt sie dort hin, so zeigt der zugehörige Eintrag wiederum auf eine bereits initialisierte Speicherstelle. Ist dies wieder die Ausgangszelle, so ist sie initialisiert, andernfalls kann sie initialisiert werden, indem ein neuer Eintrag im Feld der initialisierten Einträge eingefügt wird.

Damit kann in konstanter Zeit geprüft werden, ob eine Zelle initialisiert ist. Da die Graphen nur lineare Kantenanzahl besitzen kann damit die Adjazenzmatrix des Graphen verwaltet werden ohne $O(n^2)$ Zeit für die Initialisierung zu benötigen.

5.6 Anwendung auf das Matching-Problem

Sei G ein zweifach zusammenhängender Graph mit Maximalgrad k. Berechnet man in einem solchen Graphen G nun zunächst einen Spannbaum mit Maximalgrad $\lceil (k+2)/2 \rceil$ und bestimmt anschließend in diesem Baum mit dem Algorithmus aus Abschnitt 3.1 ein Matching, so ergibt sich folgender Satz:

Satz 8. *Sei G ein zweifach zusammenhängender Graph mit Maximalgrad k, k sei eine Konstante. Dann besitzt G ein Matching der Größe $2(n-1)/(k+3)$. Ein solches Matching kann deterministisch in $O(n \log^4 n)$ Zeit gefunden werden.*

Beweis. Die Existenz des Matchings folgt direkt aus der Existenz eines Spannbaums mit Maximalgrad $\lceil (k+2)/2 \rceil \leq (k+3)/2$ und Satz 2 aus Kapitel 3. Die Laufzeit für das Auffinden des Matchings im Spannbaum ist linear. Die Gesamtlaufzeit wird also durch die Laufzeit für das Berechnen des Spannbaums dominiert. □

Im Fall von planaren Graphen lässt sich die Laufzeit mit Bemerkung 1 auf $O(n \log^2 n)$ Zeit bei linearem Speicherbedarf und auf $O(n \log n)$ Zeit bei quadratischem Speicherbedarf verbessern.

5.7 d-facher Zusammenhang

Die Schranke für die Existenz von Matchings in Graphen mit Maximalgrad k lässt sich weiter verschärfen, wenn man stärkeren Zusammenhang fordert. Strothmann zeigt weiter folgenden Satz:

Satz 9 ([Str97])**.** *Jeder d-fach zusammenhängende Graph mit Maximalgrad k besitzt einen Spannbaum mit Maximalgrad höchstens $\lceil (k+2d-2)/d \rceil$.*

Mit dem Algorithmus aus Kapitel 3 kann in diesem Spannbaum dann ein größtes Matching berechnet werden:

Korollar 3. *Sei G ein d-fach zusammenhängender Graph mit Maximalgrad k. Dann besitzt G ein Matching M mit $|M| \geq d(n-1)/(k+3d-2)$.*

Beweis. Nach Satz 9 besitzt G einen Spannbaum mit Maximalgrad höchstens $\lceil (k+2d-2)/d \rceil \leq (k+3d-2)/d$. Daher findet Algorithmus 1 aus Kapitel 3 darin ein Matching der Größe mindestens $d(n-1)/(k+3d-2)$. □

Ein solcher Spannbaum kann in Polynomialzeit mit dem Algorithmus von Fürer und Raghavachari [FR92] gefunden werden. Die Laufzeit ist aber schlechter als die des Algorithmus von Micali und Vazirani [MV80], der ein größtes Matching berechnet. Daher ist nur die Aussage zur Existenz interessant.

Ist der Graph planar, so gibt es bessere Schranken, die neben dem Zusammenhang auch den Minimalgrad mit einbeziehen [NB79]. Algorithmen für vierfach zusammenhängende planare Graphen werden in Kapitel 7 vorgestellt. Kapitel 8 befasst sich mit dreifach zusammenhängenden planaren Graphen.

6 Graphen mit Maximalgrad 3

Dieses Kapitel beschäftigt sich mit einer speziellen Klasse von Graphen mit beschränktem Maximalgrad, nämlich Graphen, deren Maximalgrad 3 ist. In diesen Graphen sind die bisher angegebenen Verfahren und Schranken nicht sehr gut. Ziel dieses Kapitels ist es ein Verfahren anzugeben, das in einem Graphen G mit n Knoten, n_2 Knoten vom Grad 2 und ℓ_2 Blättern im 2-Blockbaum ein Matching der Größe $(3n - n_2 - 2\ell_2)/3$ findet. Dies entspricht genau der von Biedl et al. angegebenen Schranke [BDD+04]. Sie geben auch Beispiele von Graphen für die diese Schranke scharf ist. Allerdings verwenden diese Beispiele keine oder nur wenige Knoten mit Grad 2. Sie fragten daher nach Beispielen mit einer nennenswerten Anzahl von Knoten mit Grad 2. Im nächsten Abschnitt wird gezeigt, dass diese Schranke auch dann scharf ist, wenn etwa ein Drittel aller Knoten Grad 2 hat. Der angegebene Algorithmus ist also optimal in dem Sinne, dass kein Algorithmus auf dieser Graphenklasse eine bessere Schranke garantieren kann.

6.1 Eine scharfe Schranke

In Graphen mit Maximalgrad 3, mit n Knoten, davon n_2 Knoten vom Grad 2 und ℓ_2 Blättern im 2-Blockbaum existiert stets ein Matching der Größe $(3n - n_2 - 2\ell_2)/6$. Biedl et al. fragen, ob es Beispielgraphen gibt, die diese Schranke erreichen und eine signifikante Anzahl von Knoten vom Grad 2 enthalten [BDD+04]. Hier wird nun eine Familie von Graphen vorgestellt, für die diese Schranke scharf ist und bei denen zirka ein Drittel der Knoten Grad 2 hat. Da es sich bei den Graphen um Bäume handelt, ist die Anzahl der Blätter im 2-Blockbaum ℓ_2 genau die Anzahl der Knoten mit Grad 1. Abbildung 6.1 zeigt einen Graphen $G_{\min}$, für den die Schranke $(3n - n_2 - 2\ell)/6 = (3 \cdot 10 - 2 - 2 \cdot 5)/6 = 3$ scharf ist. Die hervorgehobenen Kanten bilden ein größtes Matching. Der Knoten w wird als *Wurzel* des Graphen $G_{\min}$ bezeichnet. Das größte Matching $M_{\min}$ kann so gewählt werden, dass die Wurzel w frei ist.

Zwei dieser Graphen können zu einem größeren Beispiel, für das die Schranke scharf ist, zusammengesetzt werden. Dazu werden zwei identische Kopien $G_{\min}$ und $G'_{\min}$ mit Wurzeln w und w' mit zwei neuen Knoten und drei neuen Kanten wie in Abbildung 6.2 zusammengesetzt. Der so entstandene Graph besitzt eine neue Wurzel v. Die Kardinalität eines größten Matchings in diesem Graphen ist $|M| = 2|M_{\min}| + 1$. Das entspricht genau der Schranke von Biedl et al. Das Matching in Abbildung 6.2 ist gerade das Matching, das Algorithmus 1 aus Kapitel 3 mit v als Wurzel findet, wenn bei der Tiefensuche immer zuerst das

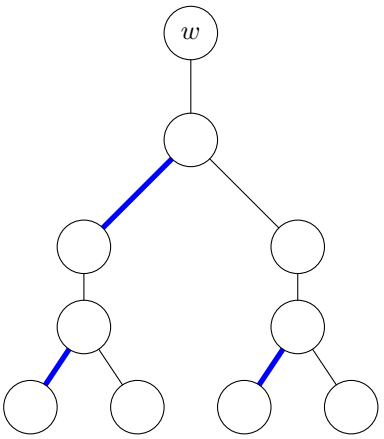


Abbildung 6.1: Graph mit zehn Knoten, fünf Blättern, zwei Knoten vom Grad 2 und einem größten Matching der Kardinalität 3.

linke Kind besucht wird. Daher ist klar, dass es sich tatsächlich um ein größtes Matching handelt.

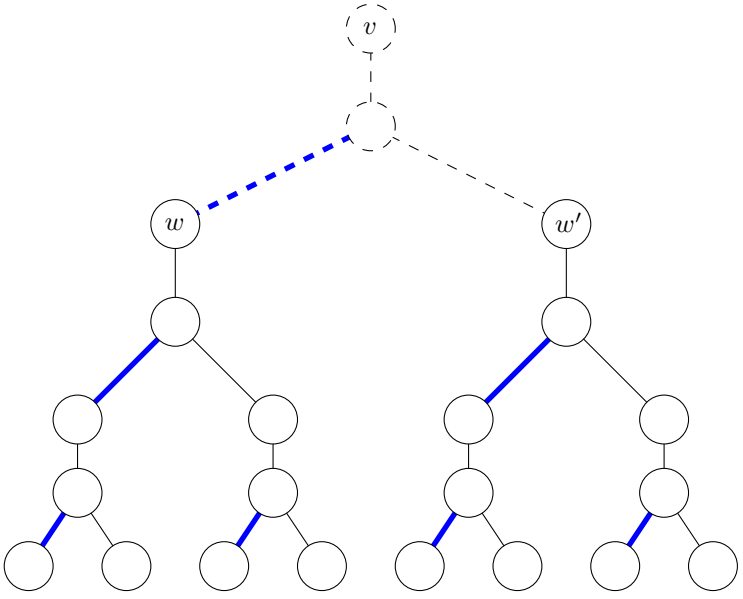


Abbildung 6.2: Zusammensetzen zweier kleiner Beispiele zu einem größeren.

Diese Konstruktion lässt sich verallgemeinern: Sei G ein solcher Graph mit Wurzel w, für den obige Schranke scharf ist. Abbildung 6.3 zeigt, wie man G um einen Graphen $G_{\min}$ und zwei weitere Knoten vergrößern kann und ein weiteres Beispiel erhält, für das die Schranke scharf ist. Sei n' die Anzahl der Knoten von G, n'_2 die Anzahl der Knoten mit Grad 2 in G und ℓ' die Anzahl der Blätter von G. Da wir annehmen, dass G die Schranke von Biedl et al. erreicht, gilt also $|M'| = (3n' - n'_2 - 2\ell')/6$ für jedes größte Matching M' in G. Für die Anzahl der Knoten n im neu konstruierten Graphen gilt: $n = n' + 12$. Die Anzahl der Knoten vom Grad 2 ist $n_2 = n'_2 + 4$, die Anzahl der Blätter $\ell = \ell' + 4$.

Ein größtes Matching in diesem Graphen besitzt Kardinalität $|M| = |M'| + 4$. Das Matching aus Abbildung 6.3 entspricht wieder dem Matching, das der Algorithmus aus Kapitel 1 mit v als Wurzel findet und ist daher größtmöglich.

Für diesen neuen Graphen ist die Schranke $(3n - n_2 - 2\ell)/6$ wiederum scharf: $(3n-n_2-2\ell)/6 = (3(n'+12)-(n'+4)-2(\ell'+4))/6 = (3n'-n'-2\ell')/6+24/6 = |M'| + 4$.

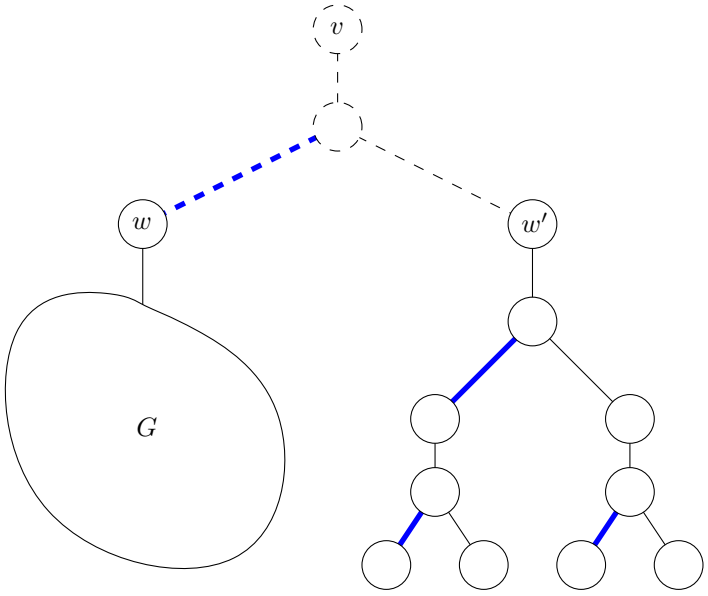


Abbildung 6.3: Vergrößern eines Gegenbeispiels um zwölf Knoten

Sei $G_0 := G_{\min}$ und G_n der Graph, der durch Anhängen von $G_{\min}$ an G_{n-1} entsteht. Alle Graphen dieser Familie erreichen die Schranke von Biedl et al. Die Anzahl der Knoten des Graphen G_n ist dabei $12 \cdot n + 10$, die Anzahl der Knoten mit Grad 2 im Graphen G_n ist $4 \cdot n + 2$. Wegen

$$\lim_{n\to\infty} \frac{4 \cdot n + 2}{12 \cdot n + 10} = 1/3$$

folgt, dass jeweils zirka ein Drittel aller Knoten Grad 2 besitzt. Die Schranke ist also optimal im folgenden Sinn: Es ist nicht möglich die Schranke zu verbessern, ohne dabei eine große Anzahl von Knoten mit Grad 2 vorauszusetzen.

6.2 3-reguläre Graphen ohne Brücken

Ein 3-regulärer Graph G ist ein Graph, in dem jeder Knoten Grad 3 besitzt. Eine Brücke ist eine Kante, nach deren Entfernung ein Graph in zwei Komponenten zerfällt. Biedl et al. [BBDL01] geben ein Verfahren an, um in einem 3-regulären Graphen, der keine Brücken enthält, ein perfektes Matching zu finden. Die Laufzeit des Algorithmus beträgt $O(n \log^4(n))$. Im Falle von planaren Graphen geben sie sogar einen Algorithmus an, der perfekte Matchings in Linearzeit findet. Da im Folgenden der Algorithmus für nichtplanare Graphen eine wichtige Rolle spielt, wird er hier vorgestellt.

Das Verfahren setzt nicht voraus, dass der Graph schlicht ist, es sind also auch Graphen mit Mehrfachkanten als Eingabe gestattet. Weiter muss der Graph nicht zusammenhängend sein. Petersen [Pet91] zeigte schon 1891, dass ein 3-regulärer Graph ohne Brücken ein perfektes Matching besitzt. Im Basisfall ist jede Kante eine Dreifachkante. Dann kann einfach eine der 3 Kanten in jeder Komponente gewählt werden, um ein perfektes Matching zu erhalten.

Besteht der Graph nicht nur aus Dreifachkanten, so existiert eine Kante, die keine Dreifachkante ist. Da der Graph 3-regulär ist, ist entweder diese Kante oder eine zu ihr inzidente Kante eine einfache Kante $e = vw$. Diese Kante heißt *Reduktionskante*. Seien a, b und w die drei Nachbarn von v und c, d und v die drei Nachbarn von w. Die Situation ist in Abbildung 6.4 dargestellt. Da e einfach ist, gilt $a, b \neq w$ und $c, d \neq v$. Einige der Knoten a, b, c und d können gleich sein.

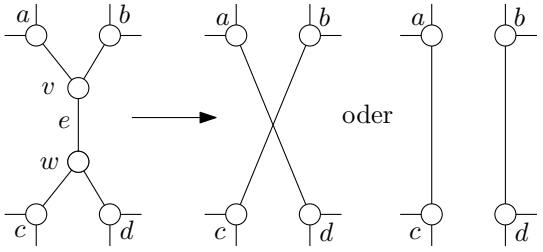


Abbildung 6.4: Gerade- und Über-Kreuz-Reduktion

Der Graph kann reduziert werden, indem die beiden Knoten v und w entfernt werden und dafür zwei neue Kanten eingefügt werden, die a, b, c und d verbinden. Dabei gibt es zwei Möglichkeiten: Entweder können a mit c und b mit d verbunden werden, diese Reduktion heißt *gerade Reduktion*. Oder es können a mit d und b mit c verbunden werden, was wir als *Über-Kreuz-Reduktion* bezeichnen. Die beiden Reduktionen sind in Abbildung 6.4 zu sehen.

Ein Lemma von König [Kőn36] besagt, dass eine dieser beiden Reduktionen zu einem Graphen ohne Brücken mit der gleichen Anzahl an Zusammenhangskomponenten wie im Originalgraphen führt. Wird also die richtige Reduktion gewählt, so besitzt der entstehende Graph nach dem Satz von Petersen wiederum ein perfektes Matching. Dieses Matching kann zu einem Matching im Originalgraphen erweitert werden:

- Ist keine der beiden Reduktionskanten im Matching enthalten, so kann M durch Hinzunahme von vw zu einem perfekten Matching im Originalgraphen erweitert werden, siehe Abbildung 6.5.
- Ist genau eine der Reduktionskanten im Matching enthalten, so muss diese aus M entfernt werden, da sie im Originalgraphen nicht vorhanden war. Dadurch werden die beiden Endknoten der Reduktionskante frei und können wie in Abbildung 6.6 zu v und w gematcht werden.
- Sind beide reduzierten Kanten im Matching enthalten, so existiert ein alternierender Kreis, der eine der reduzierten Kanten enthält [Kőn36]. Durch Umdrehen des alternierenden Kreises entsteht ein neues perfektes

Matching, das entweder eine oder beide reduzierten Kanten nicht mehr enthält.

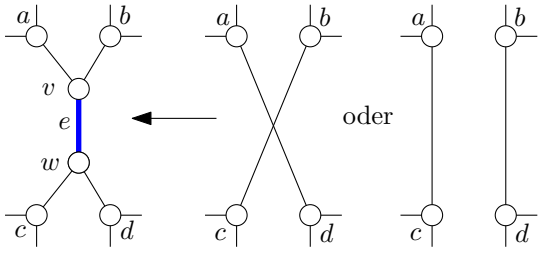


Abbildung 6.5: Erweiterung des Matchings, wenn keine Reduktionskante im Matching ist.

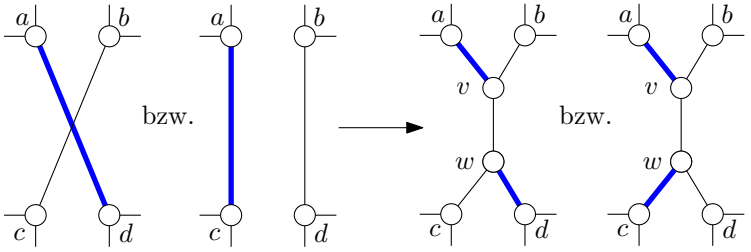


Abbildung 6.6: Erweiterung des Matchings, wenn genau eine Reduktionskante im Matching ist.

Eine direkte Umsetzung dieses Verfahrens führt zu einem Algorithmus mit quadratischer Laufzeit. Aber es geht auch schneller: Biedl et al. zeigen, dass das Matching direkt so konstruiert werden kann, dass eine fest gegebene Kante *nicht* verwendet wird. Dadurch wird das Suchen eines alternierenden Kreises vermieden, da das Matching im reduzierten Graphen direkt so konstruiert wird, dass eine der reduzierten Kanten nicht im Matching enthalten ist. Um zu entscheiden, welche der beiden Reduktionsmöglichkeiten, gerade oder Über-Kreuz-Reduktion, die richtige Wahl ist, verwenden sie eine dynamische Datenstruktur für den zweifachen Kantenzusammenhang von Holmet et al. [HdLT01]. Diese Datenstruktur erlaubt Einfügen und Löschen von Kanten in einen Graphen, sowie Anfragen, ob zwei Knoten durch zwei kantendisjunkte Pfade miteinander verbunden sind, in $O(\log^4 n)$ Worst-Case-Zeit. Die Wahl der richtigen Reduktion kann damit durch Ausprobieren in $O(\log^4 n)$ Zeit pro Reduktionsschritt durchgeführt werden: Versuche zunächst die gerade Reduktion. Dann kann in $O(\log^4 n)$ Zeit festgestellt werden, ob der resultierende Graph noch zweifach zusammenhängend ist. Ist dies nicht der Fall, so wird die Reduktion rückgängig gemacht, und wir wissen, dass die Über-Kreuz-Reduktion zu einem zweifach zusammenhängenden Graphen führt.

Dadurch ergibt sich ein Algorithmus, der in $O(n \log^4 n)$ Zeit in einem 3-regulären Graphen, der keine Brücken enthält, ein perfektes Matching findet. Dabei kann zu Beginn eine Kante angegeben werden, die im konstruierten Matching nicht enthalten sein soll.

Weiterhin zeigen Biedl et al., dass es sogar möglich ist *zwei* Kanten $e_1^{\mathrm{NM}}, e_2^{\mathrm{NM}}$ anzugeben, die nicht im konstruierten Matching enthalten sein sollen. Dazu werden die beiden Kanten durch je einen Knoten unterteilt und es wird eine neue Kante e_{M} eingefügt, die die beiden neuen Knoten miteinander verbindet, siehe Abbildung 6.7. Nun wird ein perfektes Matching in diesem Graphen konstruiert und, nötigenfalls durch Umdrehen eines Kreises, der e_M enthält, dafür gesorgt, dass e_{M} in diesem Matching enthalten ist. Dieses Umdrehen ist in Linearzeit möglich [BBDL01]. Dadurch sind die Kanten, die durch Unterteilung von e_1^{NM} und e_2^{NM} entstanden sind, nicht im Matching enthalten. Entfernen der hinzugefügten Knoten führt zu einem perfekten Matching von G, das e_1^{NM} und e_2^{NM} nicht enthält.

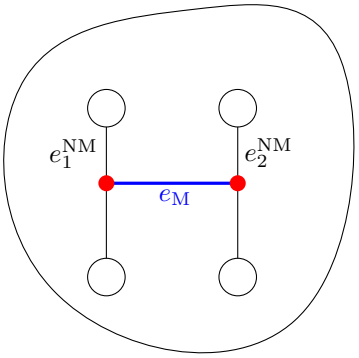


Abbildung 6.7: Konstruktion eines Matchings, das zwei Kanten e_1^{NM} und e_2^{NM} nicht enthält.

Mit dieser Verbesserung ist es möglich in einer etwas größeren Graphenklasse perfekte Matchings in $O(n \log^4 n)$ Zeit zu konstruieren.

In diesem Kapitel wird ein Begriff ähnlich dem 2-Blockbaum verwendet. Allerdings ist dieser Fall einfacher, da keine eigenen Knoten für Separatorknoten nötig sind, denn jeder Separatorknoten ist auch zu einer Brücke inzident. Der *Blockbaum* eines Graphen ist der Baum der zweifach kantenzusammenhängenden Komponenten, dessen Kanten die Brücken des Graphen sind. Da für Graphen mit Maximalgrad 3 die Begriffe zweifacher Knotenzusammenhang und zweifacher Kantenzusammenhang identisch sind, unterscheidet sich dieser Baum hinsichtlich der Zahl der Blätter nicht von der anderen Definition.

Das Theorem von Petersen besagt eigentlich, dass jeder 3-reguläre Graph G, dessen Blockbaum höchstens zwei Blätter hat, ein perfektes Matching besitzt. Ein solches Matching kann in $O(n \log^4 n)$ Zeit konstruiert werden [BBDL01]. Dazu werden zunächst alle Brücken entfernt. Dadurch entstehen in jeder Komponente des Graphen bis zu zwei Knoten mit Grad 2. Diese werden ebenfalls entfernt und die losen Ende ihrer inzidenten Kanten werden verbunden. Diese neuen Kanten werden als Nicht-Matching-Kanten gekennzeichnet. Nun wird in jeder Komponente ein perfektes Matching berechnet, das die Nicht-Matching-Kanten nicht enthält. Die Vereinigung all dieser Matchings und aller Brücken ist ein perfektes Matching in G. Das Vorgehen ist in Abbildung 6.8 dargestellt.

Biedl et al. merken an, dass das Einfügen der zusätzlichen Kante e_{M} die Planarität nicht erhält. Daher lässt sich ihr Linearzeitalgorithmus für die einzelnen

Zweifachzusammenhangskomponenten nicht direkt für planare 3-reguläre Graphen, deren Blockbaum ein Pfad ist, verwenden. Mit einem Trick ist es aber möglich trotzdem die Planarität auszunutzen:

Ist der Graph G planar und sein Blockbaum ein Pfad, so kann G wie zuvor in Komponenten zerlegt werden. Diese sind dann ebenfalls planar. Dann kann in insgesamt $O(n)$ Zeit in allen Komponenten ein perfektes Matching berechnet werden. Diese Matchings haben noch nicht ganz die nötigen Eigenschaften, um zu einem perfekten Matching des ganzen Graphen erweitert zu werden. Es ist jedoch klar, dass ein geeignetes Matching existiert. Entfernt man nun in jeder der Komponenten die beiden Kanten $e_1^{\mathrm{NM}}, e_2^{\mathrm{NM}}$, so werden dadurch in jeder Komponente höchstens vier Knoten frei. In jeder Komponente müssen also höchstens zwei augmentierende Pfad berechnet werden. Wichtig ist dabei, dass die augmentierenden Pfad lokal in den Komponenten berechnet werden können. Sei k die Anzahl der Komponenten, und n_i die Anzahl der Knoten von Komponente i, so ist insgesamt ein Aufwand von $O(\sum_{i=1}^{k} n_i \alpha(n_i))$ nötig. Es gilt $\alpha(n_i) \leq \alpha(n)$ und $\sum_{i=0}^{k} n_i \leq n$, da jeder Knoten in höchstens einer Komponente ist. Damit ergibt sich ein Gesamtaufwand von $O(n\alpha(n))$, um in planaren 3-regulären Graphen, deren 2-Blockbaum ein Pfad ist, ein perfektes Matching zu berechnen. Das folgende Lemma fasst dies zusammen:

Lemma 7. *Sei G ein planarer 3-regulärer Graph, dessen 2-Blockbaum ein Pfad ist. Dann kann ein perfektes Matching in $O(n\alpha(n))$ Zeit berechnet werden.*

Leider lässt sich diese Verbesserung für planare Graphen im Folgenden aber nicht weiterverwenden.

6.3 Beliebige 3-reguläre Graphen

Im vorigen Abschnitt wurde ein Verfahren von Biedl et al. [BBDL01] vorgestellt, mit dem in 3-regulären Graphen, deren 2-Blockbaum maximal zwei Blätter besitzt, ein perfektes Matching in $O(n \log^4 n)$ Zeit gefunden werden kann. Biedl et al. zeigen, dass ein 3-regulärer Graph, dessen 2-Blockbaum ℓ_2 Blätter besitzt, stets ein Matching der Größe $(3n - 2\ell_2)/6$ besitzt [BDD+04]. Dieser Abschnitt hat das Ziel, diese Schranke unter Verwendung des Verfahrens aus dem vorigen Abschnitt zu erreichen. Dazu wird die Aussage zunächst erneut bewiesen, allerdings ist der Beweis im Gegensatz zum Beweis von Biedl et al. konstruktiv. Anschließend wird gezeigt, wie sich der aus dem Beweis resultierende Algorithmus effizient implementieren lässt.

Sei G ein 3-regulärer Graph, dessen 2-Blockbaum ℓ_2 Blätter besitzt. Die Grundidee ist, geschickt Blätter aus dem 2-Blockbaum abzuschneiden, indem gewisse Brücken gelöscht oder ersetzt werden. Um den Graphen weiterhin 3-regulär zu halten, werden Hilfsknoten hinzugefügt. Als Basisfall dient der Algorithmus von Biedl et al. aus dem vorigen Abschnitt. Es wird eine etwas stärkere Anforderung an das Matching gestellt, nämlich dass nur Knoten, die zu einer Brücke inzident sind, frei sein dürfen.

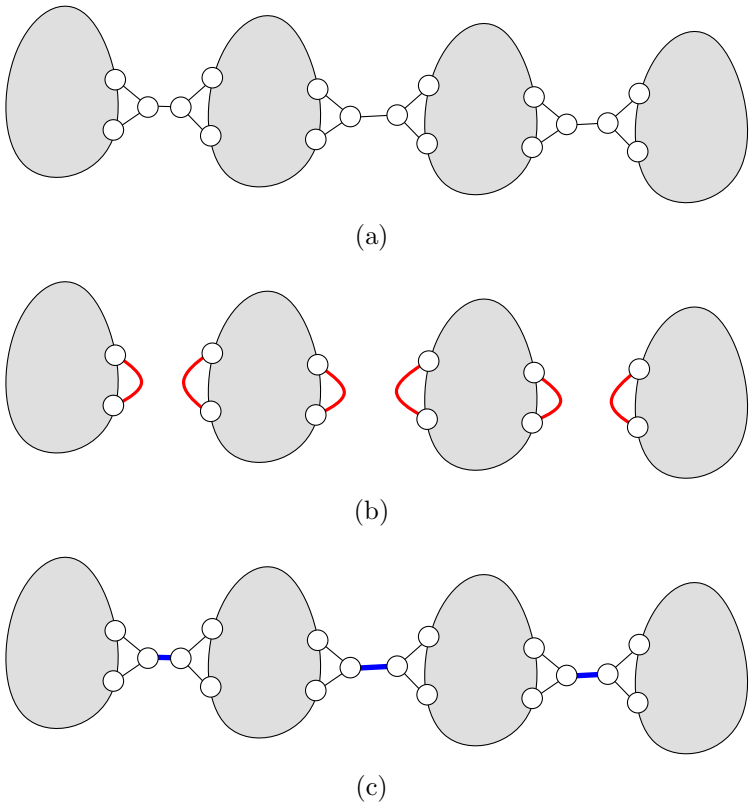


Abbildung 6.8: Schritte bei der Berechnung von perfekten Matchings in 3-regulären Graphen, deren Blockbaum ein Pfad ist.

Satz 10. *Sei G ein 3-regulärer Graph, dessen Blockbaum ℓ_2 Blätter besitzt. Dann besitzt G ein Matching M mit $|M| \geq (3n - 2\ell_2)/6$. Ferner kann M so gewählt werden, dass jeder Knoten, der nicht zu einer Brücke inzident ist, gematcht ist.*

Beweis. Der Beweis erfolgt induktiv über die Anzahl der Blätter des Blockbaums.

Gilt $\ell_2 \leq 4$, so befinden wir uns in einem der Basisfälle:

Für $\ell_2 = 1, 2$ existiert sogar ein perfektes Matching. Dieses Matching erfüllt alle Aussagen des Satzes.

Der Fall $\ell_2 = 3$ erfordert eine Fallunterscheidung. Im ersten Fall existiert ein Knoten v, nach dessen Entfernung der Graph in drei Komponenten zerfällt. Dann lässt sich der Graph wie in Abbildung 6.9 durch Entfernen der zu v inzidenten Kanten aufschneiden und zerfällt in die dort abgebildeten Komponenten. Die drei Komponenten mit mehr als einem Knoten enthalten jeweils genau einen Knoten mit Grad 2 und können durch Hinzunahme eines Hilfsgraphen H (siehe Abbildung 6.9) 3-regulär gemacht werden. Mit dem Algorithmus von Biedl et al. kann nun in jeder der 3 Komponenten ein perfektes Matching berechnet werden. Nach Entfernen des Hilfsgraphen H ist dann in jeder Komponente nur der Knoten mit Grad 2 frei. Einer dieser Knoten kann nun noch zum Knoten v gematcht werden. Es sind also $n-2 = (3n-6)/3 = (3n-2\ell_2)/3$ Knoten gematcht. Die freien Knoten sind sämtlich zu einer Brücke inzident.

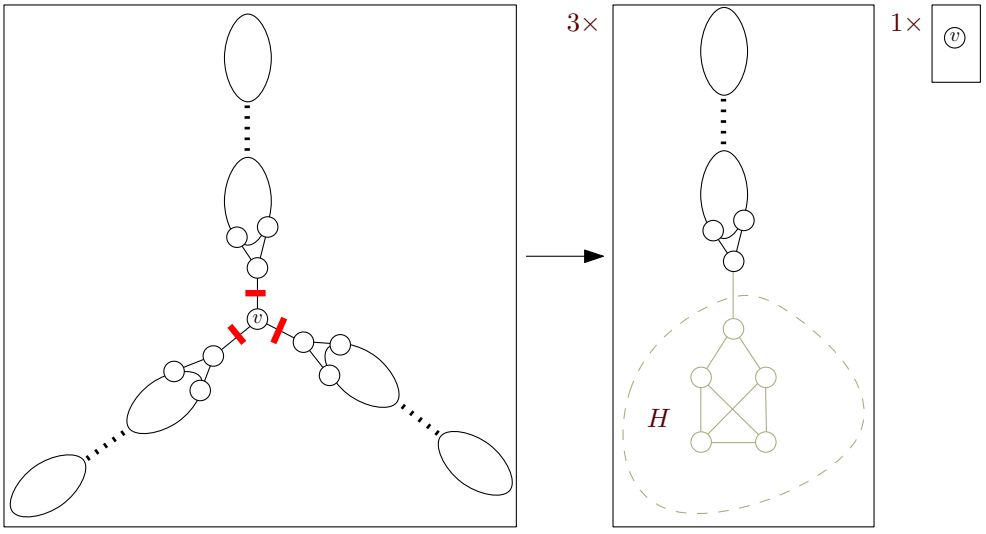


Abbildung 6.9: Abschneiden von Blättern im Fall von $\ell_2 = 3$, wenn ein Knoten v zu drei Brücken inzident ist.

Nehmen wir nun an, dass kein solcher Knoten v existiert. Beginne in einem beliebigen Blatt ℓ des Blockbaums und gehe entlang der Brücken bis zu einer Komponente, die im Blockbaum Grad 3 hat. Durch Entfernen der zuletzt benutzen Brücke zerfällt der Graph in zwei Komponenten C und C', deren Blockbäume beide Pfade sind und die jeweils einen Knoten vom Grad 2 enthalten. Diese bezeichnen wir mit $v_2(C)$ beziehungsweise $v_2(C')$. Die Komponente C', die ℓ enthält, heißt *abgeschnittenes Blatt*. Die andere Komponente C enthält noch mindestens eine weitere Brücke, die nicht zu $v_2(C)$ inzident ist, da nur ein Blatt des 2-Blockbaums entfernt wurde und bei Inzidenz zu $v_2(C)$ Fall 1 vorliegen würde. Trenne auch entlang dieser Brücke ein Blatt ab. Ähnlich wie im Fall 1 entstehen dadurch drei Komponenten, deren Blockbäume Pfade sind, siehe Abbildung 6.10. In den zwei Komponenten mit genau einem Knoten vom Grad 2 kann wie in Fall 1 durch Anfügen des Hilfsgraphen H ein Matching bestimmt werden, das nur den Knoten vom Grad 2 frei lässt. Die dritte Komponente enthält zwei Knoten vom Grad 2, u und v. Dieser Teilgraph wird durch Hinzufügen einer neuen Kante $e = uv$ 3-regulär. Im Blockbaum dieser Komponente entstehen dadurch keine zusätzlichen Blätter. In dieser Komponente kann also ein perfektes Matching bestimmt werden. Ist die Kante e nicht in diesem Matching enthalten, so ergibt sich insgesamt ein Matching mit nur zwei freien Knoten. Diese sind jeweils zu einer Brücke inzident. Ist die Kante e im Matching enthalten, so muss sie wieder entfernt werden. Die Knoten u und v können dann zu den freien Knoten in den abgeschnittenen Blättern gematcht werden. Dadurch entsteht sogar ein perfektes Matching. Die Reduktion ist in Abbildung 6.10 dargestellt.

Der letzte Basisfall ist der Fall mit $\ell_2 = 4$. Es gibt also zwei Blätter, die wie zuvor abgeschnitten werden können. Die dritte Komponente, die dabei entsteht, nennen wir im Folgenden *Hauptkomponente*. Analog wie zuvor existiert in den abgeschnittenen Blättern ein Matching, das nur einen Knoten frei lässt, nämlich den Knoten, der zur entfernten Brücke inzident ist. Die Hauptkomponente besitzt nun zwei Knoten u, v vom Grad 2 und ihr Blockbaum ist ein Pfad. Nach

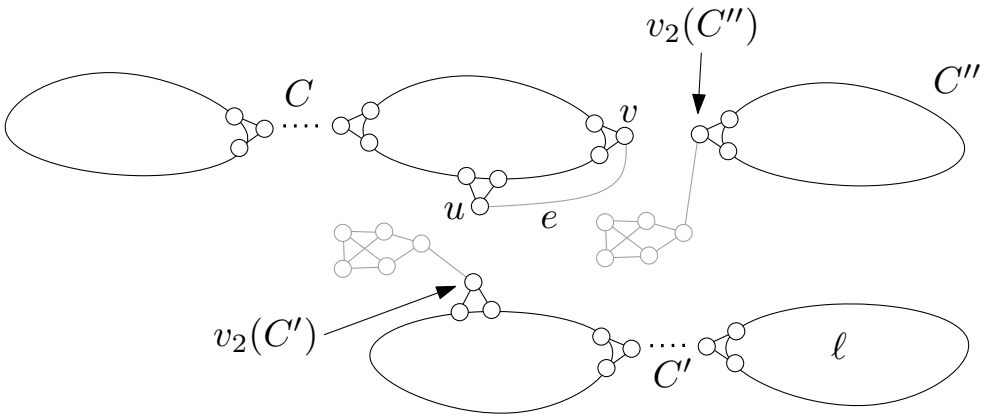


Abbildung 6.10: Abschneiden von Blättern im Fall von $\ell_2 = 3$, wenn kein Knoten zu drei Brücken inzident ist.

Hinzufügen einer zusätzlichen Kante $e = uv$ existiert in dieser Hauptkomponente also ein perfektes Matching. Nach dem Entfernen der Kante e gibt es zwei Fälle: e war im perfekten Matching enthalten, dann werden die Knoten u und v frei und können mit jeweils inzidenten freien Knoten aus den abgeschnittenen Blättern gematcht werden. In diesem Fall ist das Matching perfekt. Andernfalls bleiben die beiden Knoten aus den abgeschnittenen Blättern die einzigen freien Knoten. Diese sind beide zu Brücken inzident. Dieses Matching besitzt die Größe $(n-2)/2 \geq (3n - 2 \cdot 4)/6 = (3n - 2\ell_2)/6$.

Nun erfolgt der Induktionsschritt. Die Grundidee ist, immer geschickt drei Blätter des Blockbaums zu entfernen, so dass sich die Lösung nicht zu stark verschlechtert. Zur Einhaltung der Schranke $(3n - 2\ell_2)/6$ dürfen durch das Abschneiden von drei Blättern höchstens zwei Knoten zusätzlich frei bleiben. Zunächst ist es wichtig sicherzustellen, dass es stets möglich ist, drei Blätter zu finden, nach deren Entfernung tatsächlich auch drei Blätter weniger im Blockbaum des Restgraphen vorhanden sind. Anschließend muss sichergestellt werden, dass durch das Abschneiden der Blätter das Matching nicht zu schlecht wird.

Sei also nun $\ell_2 \geq 5$. Beginne in einem Blatt des Blockbaums und laufe entlang der Brücken bis zu einem Knoten mit Grad mindestens 3. Die zuletzt verwendete Brücke kann zum Abschneiden des Blattes verwendet werden. Abbildung 6.11 zeigt die Situation. Jedes Blatt besitzt einen Zeiger auf eine Brücke, an der es abgeschnitten werden kann, so dass die Anzahl der Blätter des Blockbaums sinkt. Besitzt der Knoten des Blockbaums, an dem abgeschnitten wird, nach Entfernen dieser Brücke immer noch mindestens Grad 3, so ist nichts weiter zu tun. Dies ist der Fall, wenn in Abbildung 6.11 etwa das Blatt ℓ zum Abschneiden gewählt wird. Wird jedoch beispielsweise das Blatt u gewählt, so sinkt der Grad von w auf 2 ab, und es existiert ein weiteres Blatt v, das zuvor an dieser Stelle abgeschnitten werden sollte. Ein Abschneiden an dieser Stelle würde aber nicht mehr zu einer Reduktion der Blätter im Blockbaum führen, da w anschließend Grad 1 besäße und damit zu einem Blatt würde. Dieses eine Blatt kann also zunächst nicht sinnvoll abgeschnitten werden und wird aus der Liste der möglichen Blätter zum Abschneiden entfernt. Darin sind nun noch immer mindestens drei Blätter enthalten. Davon kann wieder eines

beliebig gewählt werden. Analog wie zuvor wird dadurch höchstens ein Blatt unbrauchbar. Dann ist aber immer noch mindestens ein Blatt vorhanden, das abgeschnitten werden kann. Werden nun die drei gewählten Brücken entfernt, so zerfällt der Graph in vier Komponenten: Drei *Zweigkomponenten*, die jeweils ein Blatt des Blockbaums enthalten, und die *Hauptkomponente*, die drei Knoten vom Grad 2 enthält.

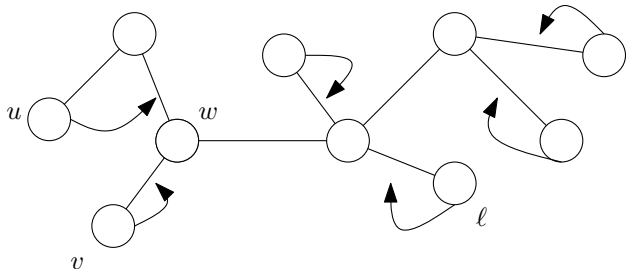


Abbildung 6.11: Abschneiden des Blattes u verbietet Abschneiden von v.

Nachdem die Existenz von drei Blättern, nach deren Entfernung die Anzahl der Blätter im 2-Blockbaum um drei sinkt, sichergestellt ist, können diese drei Blätter nun durch Entfernen der zugehörigen Brücken abgeschnitten werden. Wie zuvor kann jeder der drei abgeschnittenen Zweige mit einem Hilfsgraphen H zu einem 3-regulären Graphen erweitert werden, dessen 2-Blockbaum ein Pfad ist. In jeder der drei abgeschnittenen Komponenten existiert dann ein Matching, bei dem nur der Knoten frei ist, der zur abgeschnittenen Brücke inzident ist. Die Hauptkomponente enthält nun drei Knoten vom Grad 2. Dies wird abgefangen, indem ein neuer Knoten h vom Grad 3 hinzugefügt wird, der zu jedem der drei Knoten vom Grad 2 verbunden wird. Dadurch wird die Hauptkomponente 3-regulär. Die Situation ist in Abbildung 6.12 dargestellt. Da die Hauptkomponente schon zuvor zusammenhängend war, zerfällt sie nicht durch Entfernen einer der drei neu hinzugefügten Kanten. Der Knoten h ist also nicht zu einer Brücke inzident. Nach Induktionsvoraussetzung existiert in der Hauptkomponente ein Matching, bei dem höchstens $2(\ell_2 - 3)/3$ Knoten frei sind. Weiterhin sind diese freien Knoten sämtlich zu einer Brücke inzident. Insbesondere ist der neu hinzugefügte Knoten h nicht frei. Er muss daher zu einem der Knoten, an denen ein Blatt abgeschnitten wurde, gematcht sein. Um ein Matching im Originalgraphen zu erhalten, muss der Knoten h entfernt werden. Dadurch wird der Knoten, zu dem er gematcht war, frei und dieser kann zu dem freien Knoten im zugehörigen abgeschnittenen Zweig gematcht werden. Im resultierenden Matching sind nur $2(\ell_2 - 3)/3 + 2 = 2\ell_2/3$ Knoten frei. Diese sind nach Konstruktion und Induktionsvoraussetzung zu Brücken inzident. □

Dieser konstruktive Beweis der Schranke von Biedl et al. lässt sich nun leicht in einen Algorithmus umwandeln, der ein solches Matching tatsächlich findet. Als Basisfall wird dabei der Algorithmus aus Abschnitt 6.2 verwendet. Die genaue Implementierung und Laufzeitabschätzung wird im nächsten Kapitel vorgestellt. Anschließend folgt noch eine leichte Verallgemeinerung auf eine etwas größere Graphenklasse.

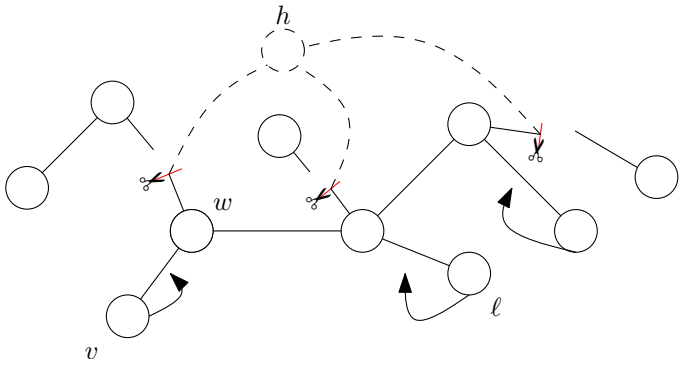


Abbildung 6.12: Abschneiden von drei Blättern und Wiederherstellung der 3-Regularität der Hauptkomponente

6.4 Implementierung

Der Beweis aus dem vorherigen Abschnitt führt direkt zu einem Verfahren um in einem 3-regulären Graphen G, dessen 2-Blockbaum ℓ_2 Blätter besitzt, ein Matching M mit $|M| \geq (3n - 2\ell_2)/6$ zu finden. Der Algorithmus lässt sich so implementieren, dass er eine Laufzeit von $O(n \log^4 n)$ Zeit benötigt.

Algorithmus 2 : MatchThreeRegularGraph

Eingabe : 3-regulärer Graph G, 2-Blockbaum $\mathcal{T}$ von G
Ausgabe : Matching M der Größe $(3n - 2\ell_2)/6$
Beginn
  wenn $\ell_2 \leq 4$ **dann**
    Berechne M wie im Beweis zu Satz 10.
    gib M **zurück**
  Finde drei Blätter und schneide sie ab.
  $G' \leftarrow$ Hauptkomponente (mit Hilfsknoten 3-regulär)
  $\mathcal{T}' \leftarrow$ Blockbaum von G'
  $(C_1, C_2, C_3) \leftarrow$ abgeschnittene Zweigkomponenten
  $M \leftarrow$ MatchThreeRegularGraph$(G', \mathcal{T}')$
  Berechne Matchings M_1, M_2, M_3 in C_1, C_2, C_3 mit dem Algorithmus von Biedl et al. [BBDL01].
  $M \leftarrow M \cup M_1 \cup M_2 \cup M_3$
  Entferne zu Hilfsknoten inzidente Kante aus M.
  Füge Kante zwischen frei gewordenem Knoten und Knoten aus Zweig zu M hinzu.
  gib M **zurück**
Ende

Zur Analyse der Laufzeit kümmern wir uns zunächst um die schnelle Bestimmung der Brücken zum Abschneiden von Blättern, sowie die Berechnung des Blockbaums der Hauptkomponente.

Lemma 8. *Gegeben einen 3-regulären Graphen G, so ist es möglich in $O(n\alpha(n))$ Gesamtzeit wiederholt drei Zweige zum Abschneiden von Blättern zu bestimmen, die Zweige zu entfernen, die Grad-2-Knoten der Hauptkomponente mit*

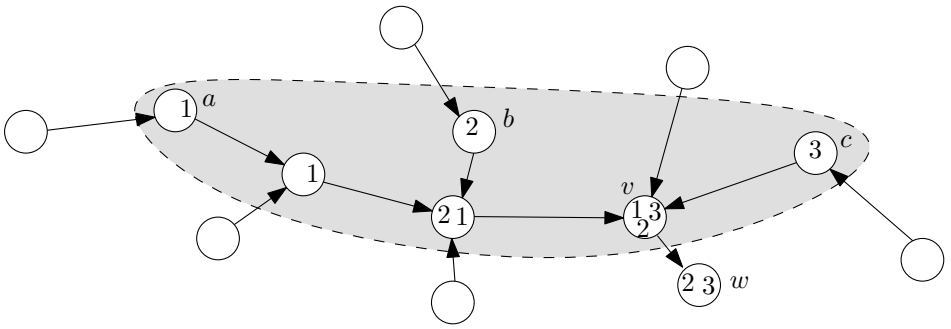


Abbildung 6.13: Kontraktion der Knoten zwischen a, b und c.

einem neuen Knoten zu verbinden und den Blockbaum $\mathcal{T}$ der Hauptkomponente zu aktualisieren. Die Prozedur stoppt, wenn $\mathcal{T}$ weniger als fünf Blätter hat.

Beweis. Zu Beginn wird der 2-Blockbaum $\mathcal{T}$ einmal berechnet. Mit dem Algorithmus von Tarjan [Tar72] benötigt dieser Schritt Linearzeit.

Zunächst wird ein beliebiger Knoten w von $\mathcal{T}$ als Wurzel bestimmt und alle Kanten zu w gerichtet. Weiter wird eine Liste $\mathcal{L}$ aller Blätter von $\mathcal{T}$ verwaltet. Zusätzlich zum Blockbaum werden die folgenden *Schnittinformationen* mitgeführt: Jedes Blatt besitzt einen Zeiger auf die Brücke, an der es abgeschnitten werden kann. Für jeden Knoten wird eine Liste aller Blätter verwaltet, die an einer eingehenden Kante abgeschnitten werden können.

Zunächst werden drei Blätter zum Abschneiden bestimmt. Wie im Beweis zu Satz 10 gesehen, können drei solche Blätter gefunden werden. Dadurch werden höchstens drei Blätter ungültig und können nicht mehr an der Stelle abgeschnitten werden, auf die sie zeigen. Diese werden markiert. Mit Hilfe der Schnittinformationen kann dies in konstanter Zeit erledigt werden. Die Knoten der Hauptkomponente, an denen abgeschnitten wird, seien a, b, c.

Da ein neuer Knoten eingefügt und zu a, b und c verbunden wird, verändert sich der Blockbaum. Es genügt nicht die abgeschnittenen Zweige zu entfernen. Zusätzlich müssen alle Knoten, die auf den eindeutigen Wegen zwischen a, b und c liegen, zu einem Superknoten kontrahiert werden. Um diese Knoten schnell zu finden, werden drei parallele Suchen bei a, b und c entlang der gerichteten Kanten gestartet. Die Suche stoppt, sobald der erste Knoten v von allen drei Suchen markiert wurde. Die Knoten, die kontrahiert werden müssen, sind alle Knoten, die von der entsprechenden Suche vor v erreicht wurden, sowie v selbst. Werden r Knoten kontrahiert, so haben die Suchen insgesamt höchstens $3r+2$ Knoten besucht. Abbildung 6.13 zeigt den Blockbaum, nachdem die parallelen Suchen durchgeführt wurden. Dabei sind Knoten, die bei der Suche mit Startknoten a, b bzw. c erreicht wurden, mit $1, 2$ bzw. 3 beschriftet. Die Menge der zu kontrahierenden Knoten ist grau hinterlegt.

Für die Kontraktionen wird eine Union-Find-Datenstruktur [Tar83] verwendet, die Kontraktionen armortisiert in $O(\alpha(n))$ Zeit ermöglicht. Daher benötigt dieser Schritt nur $O(n\alpha(n))$ Zeit während des gesamten Algorithmus.

Nachdem nun der Blockbaum der Hauptkomponente aktualisiert wurde, müssen noch die höchstens drei markierten, ungültig gewordenen Blätter aktuali-

siert werden. Sie zeigen jeweils auf eine Brücke des Graphen. Mit dieser Brücke als Ausgangspunkt gehen wir in $\mathcal{T}$ entlang der gerichteten Kanten, bis wir einen Knoten mit mindestens Grad 3 erreicht haben. Die zuletzt benutzte Kante ist dann die Stelle, an der das Blatt abgeschnitten werden kann. Da die Kanten immer nur gerichtet durchlaufen werden, wird dadurch jede Kante höchstens einmal betrachtet und dieser Schritt benötigt ebenfalls nur Linearzeit im Verlauf des ganzen Algorithmus. □

Mit Hilfe dieser Datenstruktur und dem Algorithmus von Biedl et al. [BBDL01] ist es nun leicht den Algorithmus so zu implementieren, dass er eine Laufzeit von $O(n \log^4 n)$ besitzt.

Satz 11. *In einem 3-regulären Graphen G, dessen 2-Blockbaum ℓ_2 Blätter besitzt ein Matching der Größe mindestens $(3n - 2\ell_2)/6$. Algorithmus 2 kann mit einer Laufzeit von $O(n \log^4 n)$ implementiert werden.*

Beweis. Betrachte den Graphen G. Der Blockbaum kann in $O(n)$ Zeit [Tar72] berechnet werden. Der Aufbau der initialen Datenstruktur aus Lemma 8 benötigt ebenfalls $O(n)$ Zeit. Zur Berechnung der Matchings in den abgeschnittenen Zweigen und im Basisfall wird der Algorithmus von Biedl et al. [BBDL01] mit einer Laufzeit von $O(n \log^4 n)$ verwendet.

Da für die Anzahl der Blätter des Blockbaums ℓ_2 gilt $\ell_2 \leq (n + 2)/6$ und da bei jedem Abschneiden nur eine konstante Anzahl an Hilfsknoten hinzugefügt wird, ist die Gesamtzahl der durch den Algorithmus von Biedl et al. [BBDL01] betrachteten Knoten linear. Da jeder Knoten in genau einer Komponente enthalten ist, benötigt die Berechnung aller Teilmatchings insgesamt $O(n \log^4 n)$ Zeit. Das Auffinden der geeigneten Brücken und die Aktualisierung des Blockbaums benötigt nach Lemma 8 insgesamt nur $O(n\alpha(n))$ Zeit. Daher ist die Gesamtlaufzeit des Algorithmus $O(n \log^4 n)$. □

Interessant ist die Frage, ob sich die Laufzeit für planare 3-reguläre Graphen noch verbessern lässt. Lemma 7 zeigt, wie in einem planaren 3-regulären Graph dessen Blockbaum ein Pfad ist, ein perfektes Matching in $O(n\alpha(n))$ Zeit berechnet werden kann. Es ist naheliegend zu versuchen, diesen Algorithmus zum Berechnen der Matchings in den abgeschnittenen Zweigen zu verwenden. Leider ist er nicht direkt anwendbar: Nachdem drei Zweige abgeschnitten wurden, wird die 3-Regularität der Hauptkomponente durch Einfügen eines zusätzlichen Knotens wieder hergestellt. Der dadurch entstehende Graph ist nicht notwendigerweise planar.

Um die Laufzeit zu verbessern müsste also untersucht werden, ob es möglich ist die drei Brücken zum Abschneiden stets so zu wählen, dass der Hilfsknoten mit den drei zugehörigen Kanten *planar* eingefügt werden kann. Dies ist eine offene Frage dieser Diplomarbeit.

6.5 Graphen mit Maximalgrad 3

Das zuvor vorgestellte Verfahren für 3-reguläre Graphen lässt sich auf beliebige Graphen mit Maximalgrad 3 erweitern, so dass insgesamt die von Biedl et al.

[BDD+04] gezeigte Schranke $(3n-n_2-2\ell_2)/6$ erreicht wird. Da in Abschnitt 6.1 gezeigt wurde, dass diese Schranke scharf ist, ist der Algorithmus in einem gewissen Sinne optimal.

Sei nun G ein Graph mit Maximalgrad 3 und sei n_2 die Anzahl der Knoten mit Grad 2. Wir gehen zunächst davon aus, dass alle anderen Knoten Grad 3 haben. Dann kann der Graph 3-regulär gemacht werden, indem je drei Knoten vom Grad 2 über einen zusätzlichen Knoten miteinander verbunden werden. Seien also v_1, v_2 und v_3 drei Knoten mit Grad 2. Dann kann wie in Abbildung 6.14 ein zusätzlicher Knoten h eingefügt werden, der zu v_1, v_2 und v_3 verbunden wird.

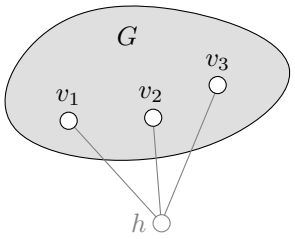


Abbildung 6.14: Herstellung der 3-Regularität bei drei Knoten vom Grad 2.

Ist n_2 durch 3 teilbar, so entsteht dadurch ein Graph G', der 3-regulär ist. Mit dem Verfahren aus diesem Kapitel kann darin also ein Matching bestimmt werden, bei dem höchstens $2\ell_2'/3$ Knoten frei sind, wobei ℓ_2' die Anzahl der Blätter des Blockbaums von G' bezeichnet. Anschließend werden die hinzugefügten Knoten und Kanten wieder entfernt. Dabei wird allerdings von je drei Knoten mit Grad 2 nur einer frei, da nur ein Knoten jedes Tripels zum Hilfsknoten gematcht sein kann. Nach Entfernung aller Hilfsknoten sind also höchstens $(2\ell_2' + n_2)/3$ aller Knoten frei. Da durch das Hinzufügen der Hilfsknoten der Blockbaum keine neuen Blätter erhält, gilt $\ell_2' \leq \ell_2$. Das Matching hat also mindestens Größe $(3n - n_2 - 2\ell_2)/6$. Dies entspricht genau der Schranke von Biedl et al. für diese Graphenklasse. Besitzt der Graph keine durch 3 teilbare Anzahl von Knoten mit Grad 2, so werden zunächst immer drei Knoten mit Grad 2 wie zuvor zusammengefasst. Dadurch entsteht ein neuer Graph, der höchstens zwei Knoten mit Grad 2 besitzt. Im Falle von $n_2 = 2$ werden beide Knoten miteinander durch eine Kante verbunden. Ist $n_2 = 1$, so kann 3-Regularität bei diesem Knoten durch den Hilfsgraphen H wie beim Abschneiden von Blättern im Beweis zu Satz 10 hergestellt werden. Dadurch ergibt sich nach Anwendung des Algorithmus für 3-reguläre Graphen und anschließendem Entfernen der hinzugefügten Kanten und Knoten ein Matching, das ebenfalls die Schranke von Biedl et al. erreicht oder höchstens um 1 darunter liegt. Dies kann durch Suche eines augmentierenden Pfades in $O(n\alpha(n))$ Zeit [Tar83] ausgeglichen werden, so dass tatsächlich die Schranke von Biedl et al. erreicht wird.

Es bleibt der Fall, dass der Graph auch Knoten vom Grad 1 besitzt. Diese werden zunächst wie in Abbildung 6.15 durch einen Hilfsgraphen H' 3-regulär gemacht. Dadurch ändert sich die Anzahl der Blätter im 2-Blockbaum nicht, da jeder Knoten mit Grad 1 ein Blatt im 2-Blockbaum ist. Mit dem vorigen Verfahren kann also ein Matching mit höchstens $(2\ell_2 - n_2)/3$ freien Knoten

berechnet werden. Da jeder freie Knoten zu einer Brücke inzident ist, muss jeder Knoten von H' gematcht sein. Dies ist nur möglich, wenn jeder Knoten von H' zu einem anderen Knoten von H' gematcht ist. Dann entstehen aber durch Entfernen von H' keine zusätzlichen freien Knoten.

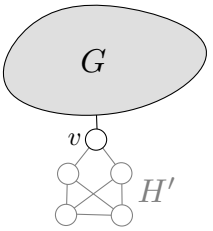


Abbildung 6.15: Herstellung der 3-Regularität bei Knoten vom Grad 1.

Insgesamt kann also mit diesem Verfahren ein Matching M mit

$$|M| \geq \frac{3n - n_2 - 2\ell_2}{6}$$

in einem beliebigen Graphen mit Maximalgrad 3 berechnet werden. Es ist klar, dass das Herstellen der 3-Regularität in Linearzeit möglich ist. Die Laufzeit des Algorithmus wird also von der Laufzeit des Verfahrens für 3-reguläre Graphen dominiert und ist daher $O(n \log^4 n)$:

Satz 12. *Sei G ein Graph mit Maximalgrad 3, n Knoten, davon n_2 mit Grad 2. Weiter habe der 2-Blockbaum von G ℓ_2 Blätter. Dann kann ein Matching der Größe $(3n - n_2 - 2\ell_2)/6$ in $O(n \log^4 n)$ Zeit berechnet werden.*

Es zeigt sich, dass das Finden dieser Matchings von der Laufzeit für das Auffinden von perfekten Matchings in 3-regulären Graphen abhängt, deren 2-Blockbaum ein Pfad ist. Der beste derzeit bekannte Algorithmus benötigt dazu $O(n \log^4 n)$ Zeit. Eine Verbesserung dieser Laufzeit würde sich direkt auf die Laufzeit des hier angegebenen Algorithmus auswirken.

7 Vierfach zusammenhängende planare Graphen

In vierfach zusammenhängenden planaren Graphen existiert stets ein Hamiltonkreis [Tut56, Tut77]. Daraus folgt bereits die Existenz eines (fast) perfekten Matchings. Chiba und Nishizeki gaben einen Algorithmus mit linearer Laufzeit zur Konstuktion eines Hamiltonkreises in solchen Graphen an [CN89].

Während also in allgemeinen Graphen ein stärkerer Zusammenhang noch zu stärkeren Aussagen über die Existenz von Matchings führen kann, ist in planaren Graphen bereits bei vierfachem Zusammenhang die Existenz eines (fast) perfekten Matchings sicher gestellt.

Im Folgenden wird das Verfahren zur Konstruktion eines Hamilton-Pfades in einem solchen Graphen vorgestellt.

7.1 Existenz von Hamiltonkreisen und interner Vierfachzusammenhang

Der Algorithmus zum Auffinden eines Hamilton-Pfades in einem vierfach zusammenhängenden Graphen basiert auf einem Divide-and-Conquer-Vorgehen. Der Graph wird in kleinere Teile zerlegt, deren Hamilton-Pfade zusammengesetzt werden können. Als Basisfall dienen Dreiecke, in denen ein Hamiltonpfad direkt angegeben werden kann.

Das Vorgehen basiert auf einem Lemma von Thomassen:

Lemma 9 ([Tho83])**.** *Sei G ein zweifach zusammenhängender planarer Graph und Z der Kreis, der die äußere Facette begrenzt. Sei s ein Knoten auf Z, $e = \{a, b\}$ eine Kante von Z und t ein von s verschiedener Knoten von G. Dann besitzt G einen Pfad P von s nach t durch e so dass gilt:*

- *jede Komponente von $G - P$ ist zu höchstens 3 Knoten von P adjazent.*
- *jede Komponente von $G - P$, die einen Knoten aus Z enthält, ist zu höchstens 2 Knoten von P adjazent.*

Dieses Lemma impliziert direkt die Existenz eines Hamiltonkreises in einem vierfach zusammenhängenden planaren Graphen G: Seien s, t zwei adjazente Knoten in Z, $e \neq \{s, t\}$ eine Kante auf Z. Betrachte nun den Pfad P aus Lemma 9, der s und t durch e verbindet. Aufgrund des Vierfachzusammenhangs von G muss aber jede Komponente von $G - P$ zu mindestens 4 Knoten

von P inzident sein. Also ist P ein Hamilton-Pfad und $P + \{s, t\}$ sogar ein Hamiltonkreis.

Das Verfahren basiert darauf, diesen Pfad von s nach t zu bestimmen. Obwohl der ursprüngliche Graph G vierfach zusammenhängend ist, entstehen bei der Zerlegung Teilgraphen, die nicht vierfach zusammenhängend sind. Sie sind jedoch *intern vierfach zusammenhängend.*

Definition 7.1. *Sei G ein zweifach zusammenhängender planarer Graph und Z der Kreis, der die äußere Facette begrenzt. Seien s und t zwei verschiedene Knoten auf Z und $e = \{a, b\}$ eine Kante von Z mit $e \neq \{s, t\}$. Durch Tauschen der Knoten s, t beziehungsweise a, b und Spiegeln der Einbettung kann ohne Einschränkung angenommen werden, dass $t \neq a, b$ und die Knoten s, a, b, t in dieser Reihenfolge im Uhrzeigersinn auf Z liegen. Achtung, $s = a$ ist möglich.*

Sei nun r der im Gegenuhrzeigersinn nächste Knoten von s auf Z und $f = \{r, s\}$ die Kante, die r und s verbindet.

Für x, y auf Z bezeichne P_{xy} den Pfad, der auf Z im Uhrzeigersinn von x nach y verläuft.

Ein zweifach zusammenhängender planarer Graph G heißt intern vierfach zusammenhängend bezüglich (s, t, e)*, wenn G folgende Bedingungen erfüllt:*

- *Ist $\{x, y\}$ ein separierendes Paar, so enthält jede Komponente von $G - \{x, y\}$ mindestens einen Knoten von Z und jeder der drei Pfade P_{sa}, P_{bt} und P_{ts} enthält höchstens einen der beiden Knoten x, y.*
- *Ist $\{x, y, z\}$ ein separierendes Tripel, so enthält jede Komponente von $G - \{x, y, z\}$ mindestens einen Knoten von Z.*

Aus der ersten Bedingung folgt direkt, dass beide Knoten x, y zu Z gehören müssen und $G - \{x, y\}$ in genau zwei Komponenten zerfällt. Die zweite Bedingung impliziert, dass von jedem separierenden Tripel $\{x, y, z\}$ mindestens zwei Knoten auf Z liegen.

Der Begriff *intern vierfach zusammenhängend* ist sowohl von der Wahl der Einbettung, als auch von der Wahl von s, t und e abhängig. Abbildung 7.1 zeigt zwei Fälle, wie intern vierfach zusammenhängende Graphen aussehen können. Im grau hinterlegten Bereich können sich weitere Knoten und Kanten befinden.

7.2 Zerlegungen

Definition 7.2. *Ein separierendes Paar $\{x, y\}$ heißt* vertikal*, wenn $x \in P_{sa} - s$ und $y \in P_{br}$ oder $x = s$ und $y \in P_{bt} - t$ gilt.*

Ist $\{x, y\}$ ein nicht-vertikales Separationspaar, so muss einer der Knoten x, y zu $P_{bt} - t$ und einer zu $P_{tr} - t$ gehören. Ein solches Paar heißt horizontal.

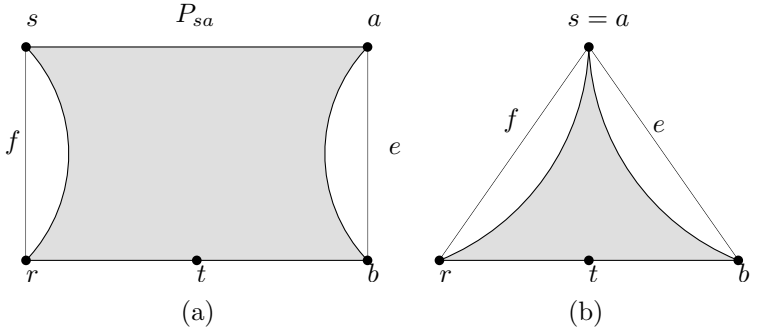


Abbildung 7.1: Intern vierfach zusammenhängender planarer Graph mit (a) $s \neq a$ und (b) $s = a$.

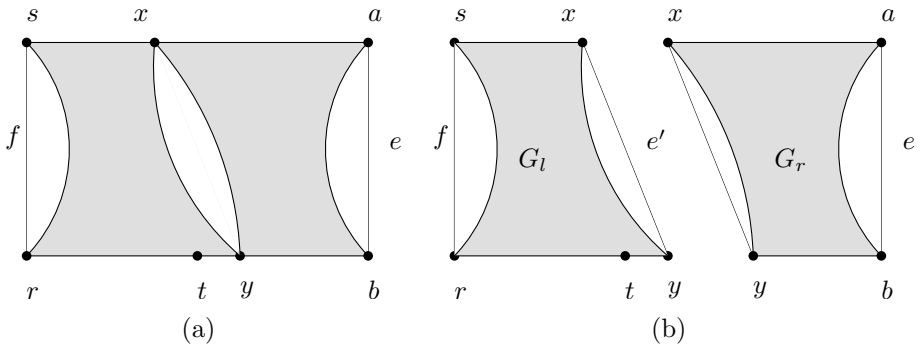


Abbildung 7.2: (a) Vertikales Separationspaar x, y, (b) Typ-I-Reduktion mit x, y.

Im Algorithmus wird G entlang den vertikalen Separationspaaren in zwei Teilgraphen zerlegt. Solch eine Zerlegung heißt dann *Typ-I-Reduktion*. Die Bedingungen für die Lage der Knoten eines vertikalen Separationspaares sorgen dafür, dass die Knoten s, t, die durch den Pfad verbunden werden sollen, bei der Zerlegung nicht in unterschiedliche Komponenten gelangen. Abbildung 7.2 (a) zeigt ein Beispiel für ein vertikales Separationspaar. Abbildung 7.2 (b) zeigt eine Typ-I-Reduktion. Dabei wird der Graph mit Hilfe des vertikalen Separationspaares x, y in zwei Komponenten G_l, G_r zerlegt. Dabei wird in beide Graphen die Kante $e' = \{x, y\}$ eingefügt. Findet man nun in G_l einen Hamilton-Pfad, der s und t verbindet und e' enthält und in G_r einen Hamilton-Pfad, der x und y verbindet und e enthält, so lassen sich die beiden Pfade zu einem Hamiltonpfad von G zusammensetzen, indem aus dem ersten Pfad e' entfernt wird und stattdessen der Pfad, der in G_r verläuft eingefügt wird.

Ist G ein intern vierfach zusammenhängender planarer Graph ohne vertikale Separationspaare, so lässt sich G in verschiedene Typen von Teilgraphen G_b, G_g^2, G_u^3 und G_g^4, die im Folgenden definiert werden, zerlegen. Bei der Zerlegung treten nicht immer alle vier Typen von Teilgraphen auf. Im Fallen von $s = a$ entsteht beispielsweise nur G_b.

Zur Konstruktion der Zerlegung sind einige Hilfsgraphen nötig. Zunächst wird aus G der Pfad P_{sa} entfernt. Sei C_b die 2-Zusammenhangskomponente von $G - P_{sa}$, die t enthält. C_b enthält dann P_{br} vollständig, da G sonst ein vertikales Separationspaar enthielte. Abbildung 7.3 zeigt Beispiele für vierfach

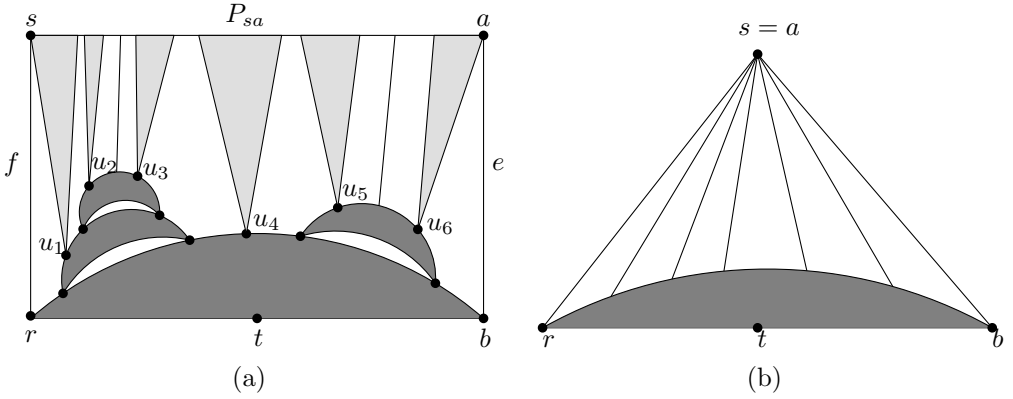


Abbildung 7.3: Zerlegung eines intern vierfach zusammenhängenden Graphen G ohne vertikales Separationspaar, (a) G mit $s \neq a$, (b) G mit $s = a$.

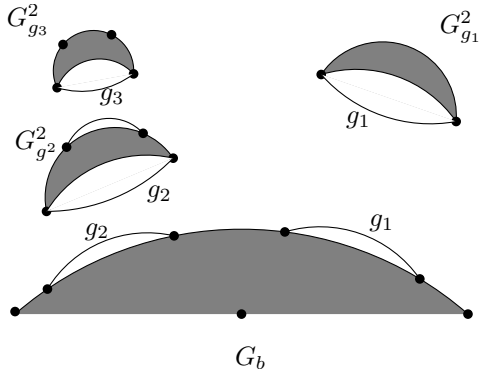


Abbildung 7.4: Die Komponenten G_b und G^2_g.

zusammenhängende Graphen G, die kein vertikales Separationspaar enthalten. Die Komponente C_b ist in beiden Graphen dunkel eingefärbt.

Nun wird C_b wiederholt an jedem Separationspaar zerlegt, so dass einer der beiden Teilgraphen den Pfad P_{br} vollständig enthält. Die Komponente, die P_{br}, enthält heißt G_b. Die anderen Komponenten von C_b heißen G^2_g, wobei $g = \{x, y\}$ eine virtuelle Kante dieser Komponente ist, also eine Kante, die zwei Knoten verbindet, die in C_b ein separierendes Paar sind. Durch Vertauschen von x und y kann $x \neq b$ angenommen werden. Ein Beispiel für die Zerlegung des Graphen aus Abbildung 7.3 (a) ist in Abbildung 7.4 dargestellt.

Als nächstes wird G^3_u für jeden Separatorknoten u von $G - P_{sa}$ in C_b definiert. C_u bezeichne den maximalen Teilgraphen von $G - P_{sa}$, der durch u von C_b abgetrennt werden kann. C^3_u bezeichne dann den Teilgraphen von G, der von den Knoten von C_u und den Knoten von P_{sa}, die zu C_u adjazent sind, induziert wird. Sei x der Knoten von $P_{sa} \cap C^3_u$, der s entlang P_{sa} am nächsten liegt. Analog sei y der Knoten von $P_{sa} \cap C^3_u$, der a entlang von P_{sa} am nächsten liegt. Es gilt $x \neq y$, sonst wäre $\{x, u\}$ ein nicht erlaubtes Separationspaar. Nun wird eine neue Kante $e' = \{u, y\}$ zu C^3_u hinzugefügt. Der so entstandene Graph heißt G^3_u. Abbildung 7.5 zeigt ein Beispiel einer solchen Komponente für den Graphen aus Abbildung 7.3.

Der letzte Typ Graph ist G^4_g für eine virtuelle Kante $g = (x, y)$. Er entsteht durch Verschmelzen aller G^2_g mit allen G^2_g und C^3_u, die zu G^2_g adjazent sind.

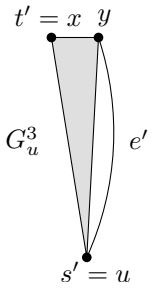


Abbildung 7.5: Komponente G_u^3

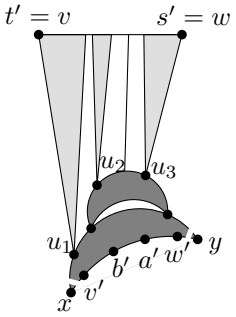


Abbildung 7.6: Komponente G_g^4

Formal lässt sich G_g^4 folgendermaßen konstruieren:

1. $C_g^4 := G_g^2$
2. Solange C_g^4 weitere virtuelle Kanten $g' \neq g$ besitzt: Verschmelze $G_{g'}$ mit C_g^4.
3. Für jeden Schnitt-Knoten u von $G - P_{sa}$ in C_g^4 verschmelze G_u^3 mit C_g^4.
4. $G_g^4 := C_g^4 - x - y$.

Außerdem sei v der Knoten von G_g^4, der auf P_{sa} am nähesten an s liegt und w der Knoten von G_g^4, der auf P_{sa} am weitesten von s entfernt ist. Zudem bezeichne w' (bzw. v') den Knoten, der zu x (bzw. y) adjazent ist und als letzter (bzw. erster) auf dem äußeren Pfad P_{wv} von G_g^4 liegt. Abbildung 7.6 zeigt eine der Komponenten G_g^4 des Graphen aus Abbildung 7.3. Die Zerlegung in diese Teilgraphen heißt *Typ-II-Reduktion.*

Folgendes Lemma besagt nun, dass alle entstehenden Teilgraphen bezüglich einer geeigneten Wahl von Knoten und Kanten intern vierfach zusammenhängend sind.

Lemma 10. *Sei G ein intern vierfach zusammenhängender Graph, der keine vertikalen Separationspaare enthält. Dann sind alle Zerlegungen G_b, G_g^2, G_u^3 und G_g^4 intern vierfach zusammenhängend bezüglich (s', t', e'), wenn s', t' und e' wie folgt definiert sind:*

1. *Fall G_b: Setze $s' = b$ und $t' = t$. Ist $t \neq r$, so sei e' die im Uhrzeigersinn nächste zu r inzidente Kante des Kreises Z', der die äußere Facette begrenzt. Ist $t = r$, so sei e' die im Gegenuhrzeigersinn nächste inzidente zu b inzidente Kante.*

2. *Fall G_g^2: Setze $s' = y, t' = t' = x$ und e' die im Uhrzeigersinn nächste zu y inzidente Kante auf dem Kreis, der die äußere Facette begrenzt.*
3. *Fall G_u^3: Setze $s' = u, t' = x$ und $e' = \{u, y\}$.*
4. *Fall G_g^4: Setze $s' = w, t' = v$. Gilt $v' \neq w'$, so sei $e' = (a, b)$ eine beliebige Kante des Pfads $P_{w'v'}$ auf dem Rand der äußeren Facette.S Ist $v' = w'$, dann sei $e' = \{a', b'\}$ eine beliebige zu w' inzidente Kante von P_{wv}.*

Beweis. Siehe [CN89]. □

7.3 Konstruktion eines Pfades

Das folgende Lemma zeigt nun, wie man mit dieser Zerlegung Hamilton-Pfade konstruieren kann.

Lemma 11 ([CN89])**.** *Sei G ein planarer Graph und Z der Kreis, der die äußere Facette begrenzt. Weiter seien s und t zwei verschiedene Knoten auf Z und $e \neq (s, t)$ eine Kante von Z. Ist G intern vierfach zusammenhängend bezüglich (s, t, e), so besitzt G einen Hamilton-Pfad $P(G, s, t, e)$, der s und t verbindet und e enthält. Außerdem gilt: Hat G kein vertikales Separationspaar, so enthält $P(G, s, t, e)$ die Kante $f = \{s, r\}$ nicht.*

Die Aussage folgt zwar schon aus Lemma 9, der Beweis ist aber konstruktiv und zeigt direkt, wie die oben definierte Zerlegung ausgenutzt werden kann. Daher wird er hier vorgestellt.

Beweis. Der Beweis erfolgt per Induktion über die Knotenanzahl n. Für $n = 3$ ist die Aussage trivialerweise erfüllt. Sei also $n > 3$. Dabei bleiben zwei Fälle zu betrachten:

Fall 1: Es existiert ein vertikales Separationspaar $\{x, y\}$. Durch Zerlegung entlang $\{x, y\}$ zerfällt G in zwei Teilgraphen G_l und G_r. Ohne Einschränkung gelte $e \in G_r$, außerdem kann $\{x, y\}$ so gewählt werden, dass G_r minimale Knotenanzahl hat. G_r besitzt dann kein weiteres vertikales Separationspaar mehr.

Sei zunächst $t \in G_l$, $e' = (x, y)$ eine virtuelle Kante. Dann ist G_l bezüglich (s, t, e') und G_r bezüglich (x, y, e) (oder (y, x, e) falls $y = b$) intern vierfach zusammenhängend. Nach Induktionsvoraussetzung existieren also Hamiltonpfade $P(G_l, s, t, e')$ und $P(G_r, x, y, e)$ (oder $P(G_r, y, x, e)$) in G_l und G_r. Diese Pfade lassen sich nach Entfernen der Kante e' zu einem Hamiltonkreis von G zusammensetzen.

Gilt $t \notin G_l$, so sei $e' = (x, y)$. Dann sind G_l und G_r intern vierfach zusammenhängend bezüglich (y, s, e') beziehungsweise (x, t, e). Es existieren also Hamiltonpfade $P(G_l, y, s, e')$, $P(G_r, x, t, e)$. Diese lassen sich nach Entfernen von e' zu einem geeigneten Hamiltonpfad von G zusammensetzen. Dabei ist wichtig dass $P(G_r, x, t, e)$ die Kante (x, y) nach Induktiosvoraussetzung nicht enthält, da G_r kein vertikales Separationspaar besitzt.

Fall 2: G besitzt kein vertikales Separationspaar. Nach Induktionsvoraussetzung und Lemma 10 besitzen alle Zerlegungskomponenten G_b, G_g^2, G_u^3, G_g^4 einen Hamiltonpfad. Diese können nun wie folgt zu einem Hamiltonkreis $P(G, s, t, e)$ von G zusammengesetzt werden:

1. Setze zunächst $P = P_{sb} + P(G_b, t, e')$.
2. Anschließend wird P zu einem Hamiltonpfad des gesamten Graphen G modifiziert.
   a) Gibt es eine virtuelle Kante $g = (x, y)$ in G_b, so modifiziere P wie folgt:

      Falls $g \in P$, setze $P := P - g + P(G_g^2, y, x, e')$. Die Kante g wird also einfach durch einen Hamiltonpfad von G_g^2 ersetzt.

      Gilt $g \notin P$, so konstruiere G_g^4 und setze $P := P - P_{vw} + P(G_g^4, w, v, e')$. Es wird also der Teilpfad P_{vw} von P durch einen Hamiltonpfad von G_g^4 ersetzt.

   b) Für jeden Schnittknoten u von $G - P_{sa}$, der zu G_b gehört, setze $P := P - P_{xy} + P(G_u^3, u, x, (u, y)) - (u, y)$. Der Teilpfad P_{xy} von P wird durch einen Hamiltonpfad von G_u^3 ersetzt. Dieser beginnt in x und endet nach Konstruktion in u und verwendet als lezte Kante (u, y). Nach Entfernen der Kante (u, y) entsteht ein Pfad, der in G_u^3 von x zu y führt und nur den Knoten u nicht besucht. Dieser wird als Knoten von G_b schon von P durchlaufen.

Der resultierende Pfad P ist ein Hamilton-Pfad $P(G, s, t, e)$ von ganz G. □

Die Konstruktion im Beweis von Lemma 11 führt direkt zu einem Algorithmus, um einen Hamiltonpfad $P(G, s, t, e)$ in einem intern vierfach zusammenhängenden planaren Graphen G zu berechnen. Die Reduktion mit einem Separationspaar wie in Fall 1 heißt dabei *Typ-I-Reduktion*, während die Zerlegung im Fall 2 *Typ-II-Reduktion* heißt. Der Algorithmus sieht dann folgendermaßen aus:

Algorithmus 3 : HPATH(G,s,t,e)

```
wenn G enthält genau 3 Knoten dann
  | gib Hamiltonpfad P(G, s, t, e) zurück ;  /* G ist ein Dreieck */
sonst
  | wenn G besitzt vertikales Separationspaar dann
  |  | Typ-I-Reduktion
  | sonst
  |  ⌊ Typ-II-Reduktion
  ⌊
```

Es ist klar, dass die Laufzeit durch den Aufwand für die Zerlegung dominiert wird. Diese Zerlegung kann in $O(n)$ Zeit durchgeführt werden [CN89]. Für die Anzahl der Reduktionsoperationen gilt folgendes Lemma:

Lemma 12. *Besitzt der Eingabegraph G für den Algorithmus HPATH n Knoten, so gibt es während der Ausführung von HPATH höchstens $n - 3$ Reduktionen.*

Beweis. Sei r die Anzahl der Reduktionsoperationen während einer Ausführung von HPATH. Sei $d(i), 1 \leq i \leq r$, die Anzahl der kleineren Graphen, in die der Graph bei der i-ten Reduktion zerlegt wird, $d(i)$ bezeichnet also die Anzahl der rekursiven Aufrufe bei Reduktion i. Ist die i-te Reduktion eine Typ-I-Reduktion, so gilt $d(i) = 2$. Ist die i-te Reduktion eine Typ-II-Reduktion und wird keiner der Graphen G_g^2, G_g^4 und G_u^3 konstruiert, so gilt $d(i) = 1$. Insgesamt wird G in Dreiecke zerlegt, für die dann Hamiltonpfade bestimmt werden. Sei t die Gesamtzahl der Dreiecke, in die G zerlegt wird.

Nun wird der sogenannte Rekursionsbaum betrachtet. Jeder innere Knoten des Baums entspricht einer Reduktion, seine Kinder entsprechen den rekursiven Aufrufen bei dieser Reduktion, und die Blätter entsprechen den endgültigen Dreiecken. In einem Binärbaum ist die Anzahl der inneren Knoten um eins geringer als die Anzahl der Blätter. Der Rekursionsbaum besitzt auch innere Knoten höheren Grades und Knoten mit Ausgangsgrad 1, die aber keine zusätzlichen Blätter produzieren. Der Baum hat t Blätter und r innere Knoten von denen r' Ausgangsgrad 1 haben. Es gilt also

$$r \leq t - 1 + r'.$$

Als nächstes wird die Länge der ingsesamt konstruierten Pfade abgeschätzt. Diese ändert sich, je nachdem in wieviele Teile der Graph bei der i-ten Reduktion zerlegt wird. Die Länge vergrößert sich bei der i-ten Zerlegungsoperation um $d(i) - 1$. Gilt $d(i) = 1$, so verkürzt sich die Länge des zu findenden Pfades um mindestens 1. HPATH soll einen Hamiltonpfad der Länge $n - 1$ finden. Daher lässt sich die Gesamtlänge der gefundenen Pfade in den Dreiecken nach oben abschätzen durch

$$n - 1 + \sum_{i=1}^{r} (d(i) - 1) - r' = n - 1 + r + t - 1 - r - r' = n + t - r' - 2.$$

Da die Gesamtlänge der in den Dreiecken gefundenen Hamiltonpfade $2t$ ist, gilt:

$$2t \leq n + t - r' - 2 \Leftrightarrow t \leq n - r' - 2.$$

Mit der Abschätzung $r \leq t - 1 + r'$ ergibt sich:

$$r \leq n - 3.$$

□

Insgesamt lässt sich der Algorithmus HPATH also so implementieren, dass er in $O(n^2)$ Zeit läuft. Chiba und Nishizeki zeigen weiter, wie der Algorithmus so implementiert werden kann, dass er Laufzeit $O(n)$ besitzt [CN89]. Dazu wird die Konstruktion der Komponenten zeitlich verzögert, bis sie wirklich benötigt werden, außerdem wird die Typ-I-Reduktion so umdefiniert, dass sie

den Graphen in einem Schritt entlang sämtlicher vertikaler Separationspaare zerlegt.

Unter Verwendung dieses Algorithmus kann nun leicht ein perfektes oder fast perfektes Matching in einem vierfach zusammenhängenden planaren Graphen in $O(n)$ Zeit gefunden werden.

Korollar 4. *Sei G ein vierfach zusammenhängender planarer Graph. Dann besitzt G ein (fast) perfektes Matching. Ein solches Matching kann in Linearzeit berechnet werden.*

Im nächsten Abschnitt werden noch einige Aussagen über die Existenz von speziellen Matchings vorgestellt.

7.4 Spezielle Matchings

Sei G ein vierfach zusammenhängender planarer Graph. Ist $|V(G)|$ ungerade und $v \in V(G)$, so existiert ein Matching M mit $|M| = (n-1)/2$ und v ist frei bezüglich M. Es kann also jeder beliebige Knoten frei gelassen werden. Zur Konstruktion wird zunächst ein beliebiger Hamiltonkreis H von G berechnet. $H - v$ ist dann ein Pfad mit einer geraden Anzahl von Knoten und besitzt daher ein perfektes Matching. Dieses Matching leistet das Gewünschte.

Es ist auch möglich ein Matching zu finden, das eine gegebene Kante $e = (u, v)$ enthält. Verwende dazu $x = u, y = v$ in Lemma 9. Dann enthält der gefundene Hamiltonkreis diese Kante und das (fast) perfekte Matching kann so gewählt werden, dass die Kante e darin enthalten ist.

Weiter ist es möglich, ein Matching größter Kardinalität zu bestimmen, das vorgegebene Kanten enthält, beziehungsweise nicht enthält. Dazu wird zunächst ein Hamiltonkreis im Graphen bestimmt und daraus ein (fast) perfektes Matching bestimmt. Anschließend werden alle Kanten entfernt, die nicht im Matching enthalten sein sollen. Dadurch entstehen eventuell freie Knoten. Das Matching kann durch Finden von augmentierenden Pfaden zu einem größten Matching erweitert werden. Das Finden eines augmentierenden Pfades benötigt $O(n\alpha(n))$ Zeit [Tar83]. Wird nur eine konstante Anzahl von Kanten entfernt, so entsteht dadurch nur eine konstante Anzahl freier Knoten und ein größtes Matching kann in $O(n\alpha(n))$ Zeit gefunden werden.

Soll eine Menge von paarweise nicht adjazenten Kanten N im Matching enthalten sein, so wird zunächst wie zuvor ein perfektes Matching bestimmt. Dann werden die Kanten zusammen mit ihren inzidenten Knoten entfernt. Anschließend kann dann wie zuvor das Matching zu einem größten Matching M vergrößert werden. $M \cup N$ ist dann ein Matching von G, das alle Kanten von N enthält und größte Kardinalität unter all diesen Matchings besitzt. Ist $|N|$ durch eine Konstante beschränkt, so kann diese Berechnung wieder in $O(n\alpha(n))$ Zeit erfolgen.

Analog können auch größtmögliche Matchings gefunden werden, die eine Knotenmenge F frei lassen. Dazu wird ebenfalls ein (fast) perfektes Matching in

G besimmt. Ist $|F|$ eine Konstante, so wird nach Entfernen von F aus G nur eine konstante Anzahl von Knoten frei. Das Matching kann dann wieder in $O(n\alpha(n))$ Zeit zu einem größtmöglichen Matching, das alle Knoten aus F frei lässt, vergrößert werden.

Mit diesen Techniken können auch reine Existenzaussagen genutzt werden: Sei G ein vierfach zusammenhängender planarer Graph, $\{u, v\}$ eine Menge von Knoten und e eine Kante, die auf dem Rand einer zu v inzidenten Facette liegt. Dann existiert ein Hamiltonkreis in $G - \{u, v\}$, der e enthält [TY94]. Dies impliziert die Existenz eines Matchings M mit folgenden Eigenschaften:

- Die Knoten u, v sind frei und nur wenn $|V(G - \{u, v\})|$ ungerade ist, ist noch genau ein weiterer Knoten frei.
- Die Kante e ist im Matching M enthalten.

Mit den obigen Techniken kann ein solches Matching in $O(n\alpha(n))$ Zeit gefunden werden. Das Auffinden von Matchings, die vorgegebene Knoten frei lassen, ist in den nächsten beiden Kapiteln von zentraler Bedeutung. Daher wird es in folgendem Lemma zusammengefasst:

Lemma 13. *Sei $G = (V, E)$ ein vierfach zusammenhängender planar Graph und V' eine Menge von Knoten von G, sowie E' eine Menge von Kanten von G mit $E' \supseteq \{uv' \mid v' \in V'\}$. Sind $|E'|$ und $|V'|$ durch eine Konstante beschränkt, so kann ein größtes Matching in $G' = (V \setminus V', E \setminus E')$ in $O(n\alpha(n))$ Zeit berechnet werden.*

Es genügt jedoch nicht G' zu kennen. Auch G muss bekannt sein, da die Berechnung des Hamiltonkreises in G erfolgt.

8 Dreifach zusammenhängende planare Graphen

Biedl et al. [BDD+04] zeigten die Existenz eines Matchings der Größe $(n+4)/3$ in einem dreifach zusammenhängenden planaren Graphen, geben allerdings keinen Algorithmus zur schnellen Konstruktion eines solchen Matchings an. Sie zeigen auch eine etwas genauere Schranke, $\min\{(2n + 4 - \ell_4)/4, \lfloor n/2 \rfloor\}$, dabei bezeichnet ℓ_4 die Anzahl der Blätter des 4-Blockbaums.

Barnette [Bar66] zeigt, dass ein dreifach zusammenhängender planarer Graph stets einen Spannbaum mit Maximalgrad 3 besitzt. Strothmann [Str97] gibt einen Algorithmus an, um einen solchen Spannbaum in Linearzeit zu konstruieren.

Unter Verwendung dieses Verfahrens und des in Kapitel 3 angegebenen Algorithmus für Bäume lässt sich leicht folgender Satz beweisen.

Satz 13. *Ein dreifach zusammenhängender planarer Graph G besitzt stets ein Matching der Größe $(n-1)/3$. Ein solches Matching kann in Linearzeit berechnet werden.*

Im folgenden Abschnitt wird der Algorithmus zum Auffinden eines Spannbaums mit Maximalgrad 3 vorgestellt werden. Ein Spannbaum mit Maximalgrad k wird dabei als *k-Spannbaum* bezeichnet. Es wird gezeigt, wie ein Matching der Größe $(n+4)/3$ in $O(n\alpha(n))$ Zeit berechnet werden kann. Anschließend wird durch Betrachtung des 4-Blockbaums und ein ähnliches Vorgehen wie in Kapitel 6 ein Algorithmus angegeben, der in $O(n\alpha(n))$ Zeit ein Matching der Größe $(2n+4-6\ell_4)/4$ findet. Für triangulierte Graphen kann in der selben Zeit sogar ein Matching der Größe $(2n + 4 - 2\ell_4)/4$ gefunden werden.

8.1 3-Spannbäume in dreifach zusammenhängenden planaren Graphen

In diesem Abschnitt sei G ein dreifach zusammenhängender planarer Graph zusammen mit einer planaren Einbettung. Ein eingebetteter Graph zerlegt die Ebene in eine endliche Anzahl von Facetten. Eine dieser Facetten ist unbeschränkt, sie heißt *äußere Facette* und wird mit $\Omega(G)$ bezeichnet.

Da der Algorithmus mit der Zeit Kanten und Knoten aus dem Graphen löscht, um den 3-Spannbaum zu konstruieren, kann der dreifache Zusammenhang nicht aufrechterhalten werden. Daher wird der Algorithmus für eine etwas andere

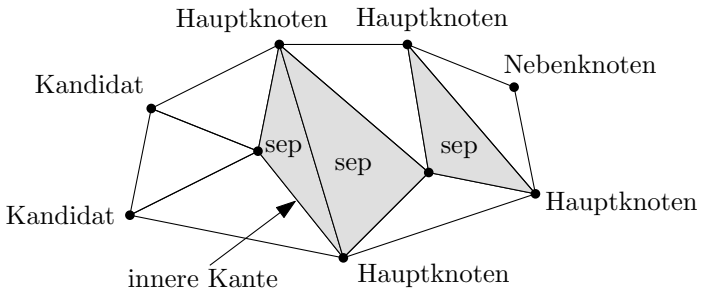


Abbildung 8.1: Beispiel eines Kreisgraphen zur Verdeutlichung der Definitionen

Graphenklasse formuliert, die jedoch eng verwandt ist: Die Klasse der Kreisgraphen.

Sei C ein einfacher Kreis in einem dreifach zusammenhängenden planaren Graphen G. G_C bezeichne den Teilgraphen von G, der von allen Kanten von C und den Kanten im Inneren von C in der Einbettung induziert wird. Jeder Graph G_C, der auf diese Art aus einem dreifach zusammenhängenden planaren Graphen G konstruiert werden kann, heißt *Kreisgraph*. Insbesondere ist jeder dreifach zusammenhängende planare Graph selbst ein Kreisgraph, da C als der Kreis gewählt werden kann, der die äußere Facette begrenzt.

Die Kanten von G_C, die nicht zu C gehören, heißen *innere Kanten*. Eine innere Kante, deren Endpunkte beide auf C liegen, heißt *Sehne*. Eine Facette f separiert G_C, wenn es zwei zu f inzidente Knoten u, v gibt, die G_C in zwei Komponenten zerlegen, so dass beide Komponenten eine innere Kante enthalten. Ein Knoten auf der äußeren Facette ist ein *Hauptknoten*, wenn er inzident zu einer inneren Kante ist, sonst heißt er *Nebenknoten*. Ein Hauptknoten, der nicht zu einer separierenden Facette inzident ist, ist ein *Kandidatenknoten*. Abbildung 8.1 zeigt einen Kreisgraphen, in dem die Begriffe verdeutlicht werden. Separierende Facetten sind grau hinterlegt.

Die Grundidee des Beweises von Barnette ist, dass Sehnen entfernt werden können, und falls keine vorhanden sind, ein Kandidatenknoten existieren muss. Ein Kandidatenknoten x lässt sich verwenden, um einen neuen Graphen zu konstruieren, der einen 3-Spannbaum besitzt, in dem x höchstens Grad 2 hat. Dadurch ist es möglich, aus diesem Spannbaum und den entfernten Kanten einen Spannbaum mit Maximalgrad 3 zu konstruieren.

Für die genauere Beschreibung sind zunächst einige grundlegende Resultate über Eigenschaften von Kreisgraphen nötig.

Es gelten folgende Aussagen [Str97]:

- Jeder Kreisgraph ist zweifach zusammenhängend.
- Ist G ein planarer Graph mit äußerer Facette $\Omega(G)$, so ist G genau dann ein Kreisgraph, wenn der Graph G^*, den man durch Hinzufügen eines neuen Knoten v und einer Kante von v zu jedem Knoten von $\Omega(G)$ erhält, dreifach zusammenhängend ist.
- Jeder Knoten eines Kreisgraphen G_C, der nicht auf C liegt hat mindestens 3 Nachbarn.

Die letzte Aussage folgt direkt aus der Definition eines Kreigraphen G_C als Teilgraph eines dreifach zusammenhängenden planaren Graphen G, da innere Knoten von G_C in G_C den gleichen Grad wie in G haben.

Die beiden folgenden Lemmata stammen von Barnette und sind die Basis für seinen Beweis und auch den Algorithmus zur Konstruktion des Spannbaumes [Bar66].

Lemma 14. *Sei G_C ein Kreisgraph und e eine Sehne. Dann ist auch $G_C - e$ ein Kreisgraph.*

Lemma 15. *Ist G_C ein Kreisgraph ohne Sehnen, so ist G_C entweder ein Kreis oder es existiert ein Kandidatenknoten in G_C.*

Der folgende Satz stammt ebenfalls von Barnette.

Satz 14. *Jeder Kreisgraph G_C besitzt einen 3-Spannbaum.*

Beweis. Der Beweis verwendet Induktion über die Anzahl der inneren Kanten von G_C. Besitzt G_C keine inneren Kanten, so ist G_C ein Kreis und besitzt damit sogar einen 2-Spannbaum. Nehmen wir nun an, der Graph G_C habe i innere Kanten und wir hätten bereits gezeigt, dass jeder Kreisgraph mit höchstens $i-1$ inneren Kanten einen 3-Spannbaum besitzt.

Wenn in G_C eine Sehne e existiert, so ist $G_C - e$ nach Lemma 14 ein Kreisgraph mit $i-1$ inneren Kanten. Er besitzt daher nach Induktionsvoraussetzung einen 3-Spannbaum T. Dieser ist aber auch ein Spannbaum von G_C.

Gibt es hingegen keine Sehne in G_C, so existiert nach Lemma 15 ein Kandidatenknoten x in G_C. Seien $\alpha_0, \ldots, \alpha_d$ die zu x inzidenten Kanten in der Reihenfolge ihrer Einbettung, wobei α_0 und α_d inzident zu Ω sein sollen. Die Endpunkte der Kante α_i seien x und x_i. F_i sei die Facette, die zu α_i und α_{i+1} inzident ist. Die Knoten der Facette F_{d-1} seien $x, x_{d-1} = y_0, y_1, \ldots, y_{e-1}, y_e = x_d$. Zudem sei y_s der erste Knoten, der zur äußeren Facette Ω inzident ist. y_{s-1} liegt also nicht auf Ω. Die Situation ist in Abbildung 8.2 dargestellt.

Sei nun $P(x, y_s)$ der Teilpfad von C von x nach y_s, der α_d nicht enthält. $P(y_0, y_s)$ bezeichne den Teilpfad des Randes von F_{d-1} von y_0 zu y_s, der α_d nicht enthält. Durch Entfernen der Nebenknoten $y_{s+1}, \ldots, y_e$ zusammen mit ihren inzidenten Kanten und den Kanten $\alpha_2, \ldots, \alpha_{d-1}$ entsteht ein neuer Kreisgraph $G'_{C'}$. Dabei besteht C' aus den Kanten $P(x, y_s) \cup P(y_0, y_s)$ und den Rändern der Facetten F_i mit $i = 1, \ldots, d-2$ ohne die Kanten $\alpha_2, \ldots, \alpha_{d-1}$. $G'_{C'}$ hat mindestens eine Kante weniger als G_C, da x ein Hauptknoten ist (also $\deg(x) \geq 3$) und daher mindestens die Kante α_2 aus G_C entfernt wurde.

Der Knoten x hat in $G'_{C'}$ Grad 2 und daher in jedem Spannbaum T' von $G'_{C'}$ Grad 1 oder 2. Daher kann jeder 3-Spannbaum T' von $G'_{C'}$ durch Hinzufügen der Knoten $y_{s+1}, \ldots, y_e$ und der Kanten $y_{s+1}y_{s+2}, \ldots, y_{e-1}y_e, y_e, x$ zu einem 3-Spannbaum von G_C erweitert werden. □

Strothmann [Str97] verwendet die Idee dieses Beweises, um einen Linearzeitalgorithmus zu entwickeln. Da der Beweis konstruktiv ist beschreibt er im

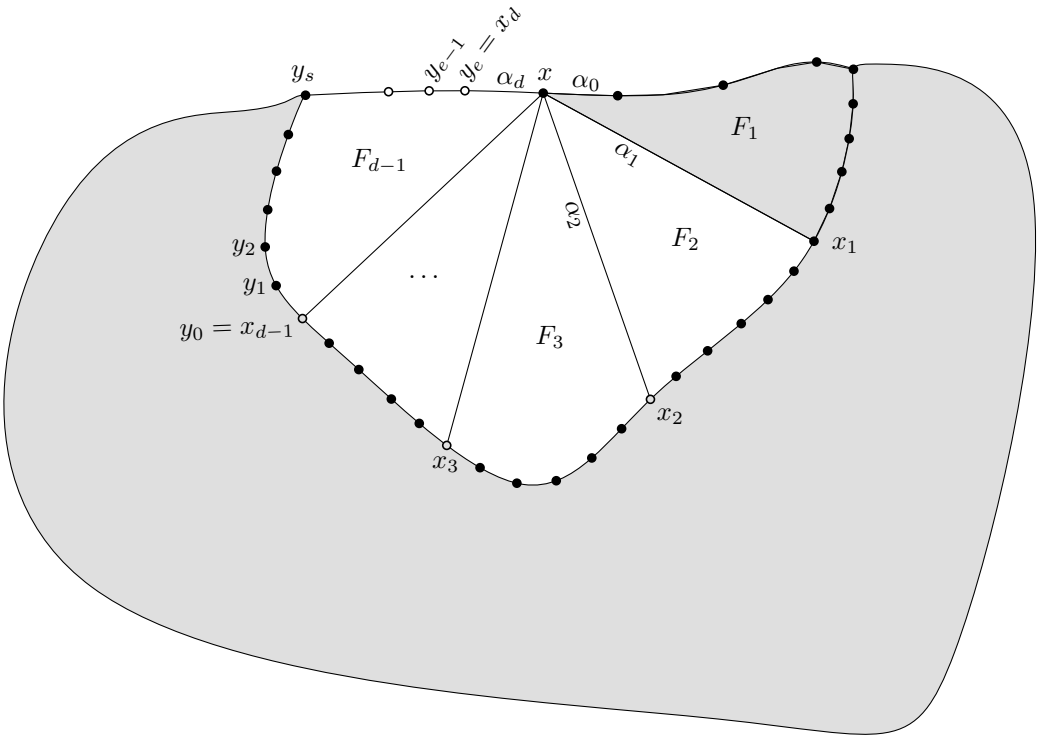


Abbildung 8.2: Löschen eines Kandidatenknoten

Prinzip bereits das Verfahren. Zunächst werden solange Sehnen entfernt, bis keine mehr existieren. Anschließend existiert nach Lemma 15 ein Kandidatenknoten. Dieser kann dann wie im Beweis zur Verkleinerung des Graphen verwendet werden.

Betrachtet man das Vorgehen zur Reduktion mittels eines Kandidatenknotens in Abbildung 8.2, so sieht man, dass alle Kanten, die betrachtet werden müssen, entweder entfernt werden, oder anschließend auf dem Rand der neuen äußeren Facette Ω liegen. Jede Kante kann aber während der gesamten Ausführung des Algorithmus höchstens einmal gelöscht und höchstens einmal neu in den Kreis, der die äußere Facette begrenzt aufgenommen werden. Da für die Reduktion für jede dieser Kanten nur konstanter Aufwand benötigt wird, benötigen die Reduktionen mittels Kandidatenknoten insgesamt nur $O(n)$ Laufzeit.

Das Entfernen von Sehnen ist schwieriger. Problematisch ist dabei, dass durch das Entfernen einer Sehne zwei Facetten vereinigt werden müssen. Da für jede Kante die Information darüber, welche Facette auf ihrer rechten Seite liegt, benötigt wird, wäre dafür eine UNION-FIND-Datenstruktur nötig und es würde mehr als $O(n)$ Zeit benötigt. Um dieses Problem zu umgehen, werden im Verfahren nur spezielle Sehnen, sogenannte *unnötige Kanten (engl. spare edges)*, entfernt. Eine unnötige Kante ist eine Sehne, die zu einer Facette inzident ist, deren sämtliche Knoten zu C gehören und die zu genau zwei Hauptknoten inzident ist. In Abbildung 8.3 ist e eine unnötige Kante.

Zudem kann die durch das Löschen neu entstehende Facette separierend sein. Abhängig davon muss die Anzahl der separierenden Facetten an den inzidenten Hauptknoten neu berechnet werden. Dadurch kann ein Hauptknoten zum Kandidatenknoten werden oder diesen Status verlieren.

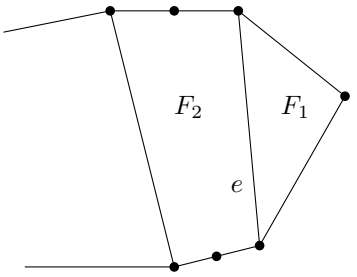


Abbildung 8.3: Löschen der unnötigen Kante e vereinigt F_1 und F_2.

Das folgende Lemma beschreibt die Bedingungen, wann ein Kreisgraph eine unnötige Kante besitzt:

Lemma 16. *Jeder Kreisgraph mit inneren Kanten, der keinen Kandidatenknoten hat, besitzt eine unnötige Kante.*

Beweis. [Str97] □

Das gesamte Verfahren sieht daher so aus:

1. Ist G_C ein Kreis, so gib einen 2-Spannbaum zurück.
2. Lösche unnötige Kanten, bis keine mehr vorhanden sind.
3. Gäbe es keinen Kandidatenknoten, so müsste nach Lemma 16 eine unnötige Kante existieren. Da aber zuvor alle entfernt wurden, existiert nun ein Kandidatenknoten x. Dieser kann zur Reduktion verwendet werden.

Um das Verfahren effizient zu implementieren müssen einige Informationen stets aktuell gehalten werden:

- Eine Liste aller Kandidatenknoten.
- Eine Liste aller unnötigen Kanten.
- Für jeden Knoten v die Anzahl der inzidenten separierenden Facetten.
- Für jede Facette die Information ob sie separierend ist oder nicht.
- Es müssen stets alle unnötigen Kanten und alle Kandidatenknoten bekannt sein.

Während des Verfahrens wird die Information benötigt, ob ein Knoten ein Kandidatenknoten ist. Es ist also nötig festzustellen, ob alle inzidenten Facetten nicht separierend sind. Dabei hilft folgendes Lemma, das charakterisiert, ob eine Facette separierend ist.

Lemma 17 ([Str97])**.** *Sei G_C ein Kreisgraph ohne Sehnen und F eine Facette in G_C. F ist separierende genau dann wenn:*

- *Es sind mindestens drei Hauptknoten zu F inzident, oder*

- *es sind zwei Hauptknoten zu F inzident und für jeden dieser Knoten v gilt:*

  Die Kanten der Facette F zu denen v inzident ist, gehören nicht zu C, liegen also nicht auf dem Rand der äußeren Facette.

Sei $e = \{u, v\}$ eine unnötige Kante und seien F_1, F_2 die zu e inzidenten Facetten. Ohne Einschränkung liegen alle zu F_1 inzidenten Knoten auf C. Wird die Kante e entfernt, so werden dadurch zwei Facetten F_1, F_2 zu einer neuen Facette F vereinigt. Dadurch kann eine neue unnötige Kante e_1 entstehen. Ist dies der Fall, so handelt es sich dabei um die nächste innere Kante von F_2. Zudem muss die Einbettung, also die Information welche Facetten inzident zu einer Kante sind, aktualisiert werden. Im Verfahren wird diese Information aber nur für Kanten benötigt, die entweder nicht auf dem äußeren Kreis C liegen oder dort zu einem Hauptknoten inzident sind. Im Algorithmus kann also einfach $F := F_2$ verwendet werden. Dadurch ist diese Information bereits für alle Kanten, die zu F_2 inzident sind, aktuell. Es müssen nur noch die zu u, v inzidenten Kanten auf dem Rand von F_1 aktualisiert werden. Das ist aber nur eine konstante Anzahl von Kanten.

Die zu F inzidenten Hauptknoten sind die Hauptknoten, die zu F_1, F_2 inzident sind. Ist ein Endknoten der Kante e nach Entfernen von e kein Hauptknoten mehr, so muss dieser aus der Liste entfernt werden. Eine Suche der Knoten in der Liste bei jedem Entfernen einer Kante wäre aber zu aufwendig. Stattdessen werden die Liste der inzidenten Hauptknoten und ihre Anzahl getrennt verwaltet. Für jede Facette wird eine Obermenge der inzidenten Hauptknoten $CM(F)$ vorgehalten, sowie die tatsächliche Anzahl an inzidenten Hauptknoten $|M(F)|$:

Für die Anzahl der zu F inzidenten Hauptknoten $|M(F)|$ gilt $|M(F)| = |M(F_1)| + |M(F_2)| - \beta$ mit

$$\beta = \begin{cases} 2 & \text{wenn } u \text{ und } v \text{ nun beide Hauptknoten sind} \\ 4 & \text{wenn nun weder } u \text{ noch } v \text{ Hauptknoten sind} \\ 3 & \text{in allen anderen Fällen.} \end{cases}$$

Es wird eine konstante Anzahl von β Operationen aufgespart, die später verwendet werden können, um Elemente aus $CM(F)$ zu entfernen.

Die Anzahl der inzidenten Hauptknoten zur neuen Facette kann also beim Entfernen einer unnötigen Kante in konstanter Zeit berechnet werden. Als Liste $CM(F)$ wird einfach die Konkatenation von $CM(F_1)$ und $CM(F_2)$ verwendet.

Die Frage, ob F separierend ist, kann nun in amortisiert konstanter Zeit beantwortet werden:

- Ist $|M(F)| = 0, 1$, so ist F nicht separierend.
- Ist $|M(F)| \geq 3$, so ist F separierend.
- Ist $|M(F)| = 2$, so sind nur 2 Hauptknoten in $CM(F)$ enthalten. Alle anderen Elemente von $CM(F)$ können mit Hilfe der für diese Liste angesparten Operationen entfernt werden. Sind die beiden Hauptknoten bekannt, so kann mit Lemma 17 in konstanter Zeit geprüft werden, ob F separierend ist.

In jedem Fall muss nur für konstant viele inzidente Knoten die Anzahl der inzidenten separierenden Facetten geändert werden. Sinkt dabei für einen dieser Knoten die Anzahl der inzidenten separierenden Facetten auf Null, so wird dieser Knoten zu einem Kandidatenknoten. Steigt die Anzahl inzidenter separierender Facetten eines Kandidatenknotens, so verliert er diesen Status und wird aus der Liste der Kandidatenknoten entfernt. Insgesamt kann also eine unnötige Kante in amortisiert konstanter Zeit entfernt werden.

Der Ablauf des Gesamtalgorithmus sieht nun wie folgt aus: Als Eingabe wird ein Kreisgraph G_C entgegengenommen. Zunächst werden die Datenstrukturen erstellt. Anschließend werden alle unnötigen Kanten entfernt. Trifft der Algorithmus dabei auf den Basisfall, einen Kreis, so wird eine beliebige Kante entfernt und der entstandene Pfad zurückgegeben. Ansonsten existiert ein Kandidatenknoten x. Dieser kann wie im Beweis von Satz 13 behandelt werden. Dabei können allerdings neue Haupt- und Nebenknoten, sowie unnötige Kanten entstehen. Daher müssen möglicherweise einige Aktualisierungen durchgeführt werden:

Für jeden Knoten v, der neu auf der äußeren Facette liegt, wird getestet, ob v ein Hauptknoten ist. In diesem Fall muss $|M(F)|$ und $CM(F)$ für die zu v inzidenten Facetten aktualisiert werden. Abhängig davon ändert sich, ob F separierend ist. Für die inzidenten Knoten ändert sich in diesem Fall die Anzahl der zu ihnen inzidenten separierenden Facetten. Die Facette F kann ihren Status wegen Lemma 17 nur ändern, wenn $|M(F)|$ konstante Größe hat. Daher muss nur für eine konstante Anzahl von Knoten die Anzahl der inzidenten separierenden Facetten aktualisiert werden. Dadurch kann die Liste der Kandidatenknoten effizient aktualisiert werden. Außerdem muss für die Knoten x, y_s geprüft werden, ob sie zu Nebenknoten wurden. Da beide nur noch zu einer Facette (außer Ω) inzident sind, geschieht die Überprüfung und Aktualisierung aller dazugehörigen Daten in konstanter Zeit.

Zuletzt müssen die neuen unnötigen Kanten bestimmt werden. Dafür kommen nur Kanten in Frage, die zu einem Knoten inzident sind, der neu auf dem Rand der äußeren Facette Ω ist. Da jeder Knoten nur einmal neu auf dem Rand sein kann, wird dabei jede Kante im gesamten Verlauf höchstens zweimal betrachtet.

Anschließend wird der entstandene Graph rekursiv weiter verarbeitet. Das Resultat wird dann durch Hinzufügen der entsprechenden Knoten und Kanten zu einem 3-Spannbaum von G_C erweitert. Der Algorithmus benötigt dafür $O(n)$ Zeit.

Wichtig ist noch die Behandlung trivialer Kanten. Eine Kante $e = \{u, v\}$ ist *trivial*, wenn beide Endknoten u und v Grad 2 haben. Diese werden einfach kontrahiert, also die Knoten u und v durch einen Knoten u/v ersetzt. Dadurch entsteht ein neuer Kreisgraph G_C^*. Jeder 3-Spannbaum T^* von G_C^* kann durch Expandieren des Knotens u/v zu einem 3-Spannbaum von G_C erweitert werden. Diese Kontraktion der trivialen Kanten wird zu Beginn einmal in Linearzeit durchgeführt. Während des Algorithmus wird bei jedem Entfernen einer Kante aus dem Graphen geprüft, ob eine der inzidenten Kanten dadurch trivial wird. Falls ja, so wird sie kontrahiert. Es ist klar, dass das in konstanter Zeit geht.

Ingesamt gilt folgender Satz:

Satz 15 ([Str97]). *Sei G_C ein Kreisgraph. Ein 3-Spannbaum von G_C lässt sich mit linearem Zeit- und Speicheraufwand konstruieren.*

Mit diesem Resultat folgt nun direkt, dass jeder dreifach zusammenhängende planare Graph G einen 3-Spannbaum besitzt und ein solcher mit linearem Zeit- und Speicheraufwand berechnet werden kann. Dazu muss der Eingabegraph zunächst mit einem der bekannten Planaritätsalgorithmen in $O(n)$ Zeit planar eingebettet werden [HT74] und eine Facette als äussere Facette Ω fixiert werden. Anschließend kann der angegebene Algorithmus verwendet werden um einen 3-Spannbaum des Graphen zu konstruieren.

Berechnet man in diesem Spannbaum mit Algorithmus 1 aus Kapitel 3 ein größtes Matching, so hat dieses nach Satz 3 mindestens Größe $(n-1)/3$.

8.2 Verbesserungen

Das im vorhergehenden Abschnitt konstruierte Matching ist nur um eine konstante Anzahl kleiner als die von Biedl et al. bewiesene Schranke, $(n+4)/3$. Eine Vergrößerung um eine konstante Anzahl von Kanten ist möglich, indem das Matching mittels augmentierender Pfade schrittweise vergrößert wird. Ein augmentierender Pfad kann in $O(n\alpha(n))$ Zeit gefunden werden [Tar83], dabei bezeichnet $\alpha(n)$ die inverse Ackermannfunktion. Damit kann die Schranke von Biedl et al. nahezu in Linearzeit erreicht werden.

8.3 Verwendung des 4-Blockbaums

Sei G ein dreifach zusammenhängender planarer Graph. Ist G nicht vierfach zusammenhängend, so existiert ein Tripel von Knoten $\{u, v, w\}$ so dass $G - \{u, v, w\}$ in Komponenten zerfällt. Ein solches Tripel heißt separierendes Tripel. Auf Grund der Planarität können durch Löschen von $\{u, v, w\}$ nur zwei Komponenten entstehen, da der Graph sonst nach dem Satz von Menger eine Unterteilung von $K_{3,3}$ enthalten würde. Sei C eine Komponente, die durch Entfernen von $\{u, v, w\}$ erhalten wurde. Daraus wird ein neuer Graph erstellt, indem zu C die Knoten u, v, w, sowie alle ihre Kanten zu adjazenten Knoten in C und die 3 Kanten uv, vw, wu, sofern sie nicht schon vorhanden waren, hinzugefügt werden.

Dieser Prozess wird iteriert, bis alle entstehenden Graphen vierfach zusammenhängend sind. Der 4-Blockbaum T von G wird nun wie folgt definiert: Es gibt einen Knoten b für jede vierfach Zusammenhangskomponente C_b des Graphen. Zwei Knoten b, b' sind genau dann verbunden, wenn es ein separierendes Tripel gibt, das sowohl zu C_b als auch zu $C_{b'}$ gehört. Der 4-Blockbaum kann in $O(n \cdot \alpha(m, n) + m)$ Zeit berechnet werden [KTBC92]. Ist G trianguliert, so ist dies sogar in $O(n)$ Zeit möglich [Kan97].

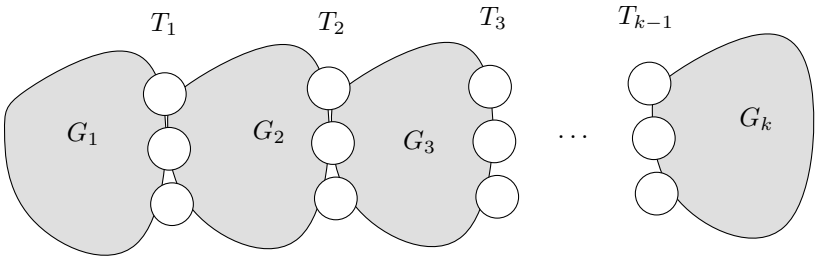


Abbildung 8.4: Vierblockbaum mit zwei Blättern.

Sei ℓ_4 die Anzahl der Blätter des 4-Blockbaums und $n := |V(G)|$. Dann existiert stets ein Matching M mit $|M| \geq \min\{(2n + 4 - \ell_4)/4, \lfloor n/2 \rfloor\}$ [BDD+04]. Ziel des restlichen Kapitels ist ein Matching ähnlicher Größe effizient zu bestimmen. Es wird ein Matching der Größe mindestens $(2n + 4 - 6\ell_4)/4$ in $O(n\alpha(n))$ Zeit konstruiert. Diese Schranke unterscheidet sich von der aus [BDD+04] lediglich um den Faktor vor ℓ_4.

Die Grundidee ist ähnlich der aus Kapitel 6 mittels Abschneiden von Blättern des 4-Blockbaums den Graphen in kleinere Teile zu zerlegen, die dann effizient verarbeitet werden können.

Ist $\ell_4 = 1$, so ist G tatsächlich vierfach zusammenhängend und es kann wie im vorigen Kapitel in Linearzeit ein (fast) perfektes Matching berechnet werden. Bereits der Fall $\ell_4 = 2$ lässt sich nicht mehr so einfach lösen. Zudem wird genau dieser Fall beim Abschneiden von Blättern benötigt. Ein schneller Algorithmus hierfür wird im nächsten Abschnitt vorgestellt.

8.4 Dreifach zusammenhängende planare Graphen mit $\ell_4 = 2$

Sei G ein dreifach zusammenhängender planarer Graph mit $\ell_4 = 2$. Die vierfach Zusammenhangskomponenten seien $G_1, \ldots, G_k$. Nun werden nacheinander geeignete Matchings in den Komponenten G_i gesucht und diese zusammengesetzt. Die Situation ist allerdings etwas schwieriger als bei 3-regulären Graphen, da die Knoten der separierenden Tripel in mehreren Komponenten vorhanden sein können. Zudem sind in den Komponenten auch zusätzliche Kanten vorhanden, die im Originalgraphen nicht vorhanden waren. Diese dürfen natürlich nicht für das Matching verwendet werden.

Sei T_i das separierende Tripel, das G_i und G_{i+1} separiert. Man beginnt mit der Berechnung eines Matchings in der Komponente G_1. Damit zu einem (fast) perfekten Matching fortgesetzt werden kann, dürfen nur Knoten in T_1 frei bleiben. Betrachtet man Komponente $G_i, i > 1$, so sind eventuell schon einige Knoten des Tripels T_{i-1} in G_{i-1} gematcht und dürfen nicht erneut gematcht werden. Zudem dürfen nur Knoten des Tripels T_i frei bleiben. Weiter darf auch keine Kante verwendet werden, die nicht schon in G vorhanden war. Es ist zu beachten, dass die sparierenden Tripel nicht disjunkt sein müssen.

Ist $|G|$ gerade, so muss ein perfektes Matching gefunden werden. Ist $|G|$ hingegen ungerade, so wähle eine Facette F der letzten Komponente, G_k, die nicht

nur durch das separierende Tripel T_{k-1} begrenzt wird und füge dort einen weiteren Knoten v ein, der zu allen Knoten der Facette verbunden wird. Dadurch bleibt der gesamte Graph dreifach zusammenhängend und planar. Demnach besitzt er ein perfektes Matching und nach Entfernen von v bleibt ein Knoten der letzten Komponente G_k frei. Es genügt folglich, in der letzten Komponente einen Knoten frei zu lassen.

In Komponente G_1 kann mit dem Verfahren aus dem vorigen Kapitel in Linearzeit ein Hamiltonkreis H berechnet werden. Ist $|G_1|$ gerade, so kann dadurch ein perfektes Matching in G_1 berechnet werden. Ist $|G_1|$ ungerade, so kann ein beliebiger Knoten aus T_1 frei gelassen werden.

Betrachte nun die Komponente $G_i, i > 1$ und sei T_{Matched} die Menge der Knoten von T_{i-1}, die bereits zuvor gematcht sind. Nachdem ein Matching in der Komponente bestimmt wurde, steht für das nächste Tripel T_i die Menge der in G_i noch freien Knoten T_{Free} fest. Leider ist unklar, ob sich jede Vorgabe von T_{Matched} so fortsetzen lässt, dass insgesamt ein (fast) perfektes Matching entsteht. Für die Frage der Fortsetzbarkeit ist nur die Vorgabe der bereits gematchten Knoten in der vorhergehenden Komponente von Bedeutung. Die Struktur des gesamten vorhergehenden Matchings hat dafür keine Bedeutung.

Eine *Matching-Konfiguration* einer Vierfachzusammenhangskomponente G_i ist ein Tupel $(T_{i,\text{Matched}}, T_{i,\text{Free}})$ mit $T_{i,\text{Matched}} \subseteq T_{i-1}$ und $T_{i,\text{Free}} \subseteq T_i$.

Eine Matching-Konfiguration von G_i heißt *realisierbar*, wenn folgende Bedingungen erfüllt sind:

- $T_{i,\text{Matched}} \cap T_{i,\text{Free}} = \emptyset$
- Es existiert ein perfektes Matching M_i in $G_i \setminus (T_{i,Matched} \cup T_{i,Free})$, das keine Kanten verwendet, die in G nicht vorhanden sind.

Die erste Bedingung besagt, dass Knoten, die in G_{i-1} bereits gematched waren in G_{i+1} nicht wieder zur Verfügung stehen. Dagegen stellt die zweite Bedingung sicher, dass sich die realisierbaren Konfigurationen verwenden lassen um ein perfektes Matching zu konstruieren.

Der *Matching-Graph* von G ist ein gerichteter azyklischer Graph, dessen Knoten genau den realisierbaren Matching-Konfigurationen der G_i entsprechen. Es können nur Matchingkonfigurationen von aufeinanderfolgenden Komponenenten miteinander durch eine Kante verbunden werden. Zwischen zwei Matchingkonfigurationen $u = (T_{i,\text{Matched}}, T_{i,\text{Free}})$ und $v = (T_{i+1,\text{Matched}}, T_{i+1,\text{Free}})$ existiert im Matchingraphen genau dann eine Kante von u nach v, wenn $T_i \setminus T_{i,\text{Free}} = T_{i+1,\text{Matched}}$ gilt. Das ist genau dann der Fall, wenn sich das in u angegebene Matching in v so fortsetzen lässt, dass dort nur die Knoten in $T_{i+1,\text{Free}}$ frei bleiben. Abbildung 8.5 zeigt einige Beispiele für Matching-Konfigurationen. Außerhalb der Komponente gematchte Knoten sind dabei durch eine Kante nach außen gekennzeichnet; Knoten, die innerhalb der Komponente gematcht sind durch eine Kante ins Innere der Komponente. Angenommen, die abgebildeten Konfigurationen seien realisierbar, so gäbe es im Matching-Graphen von G eine Kante von der Konfiguration in Abbildung 8.5 (a) nach (c), aber nicht von (a) nach (b). Die erste und letzte Komponente werden analog behandelt,

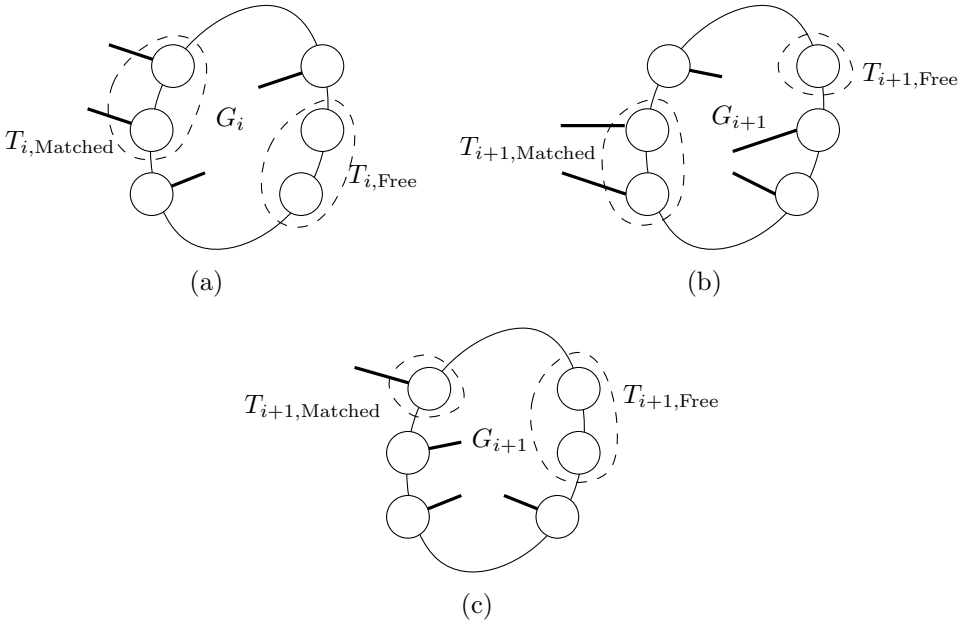


Abbildung 8.5: Einige Matchingkonfigurationen.

besitzen aber weniger Konfigurationen, da sie nur zum einem separierenden Tripel inzident sind.

Insgesamt existieren nur $O(n)$ Knoten, da es nur $O(n)$ Vierfachzusammenangskomponenten gibt und jede Komponente nur eine konstante Anzahl von realisierbaren Matching-Konfigurationen besitzt. Außerdem besitzt der Matchinggraph nur $O(n)$ Kanten. Zu jedem perfekten Matching gehört ein Pfad mit Länge $k-1$ im Matching-Graphen. Umgekehrt existiert zu jedem Pfad der Länge $k-1$ ein perfektes Matching: Dieser entspricht genau einer Abfolge von realisierbaren Matching-Konfigurationen, die so zusammenpassen, dass sich die zugehörigen Matchings M_i zu einem perfekten Matching M vereinigen lassen.

Ein perfektes Matching kann also gefunden werden, indem ein Pfad mit Länge $k-1$ im Matchinggraphen gesucht wird. Ein solcher Pfad kann in $O(n)$ Zeit gefunden werden: Beginne in allen Knoten, die zu realisierbaren Konfigurationen von G_1 gehören und beginne von dort aus eine Breitensuche. Stoppe, sobald ein Pfad der Länge $k-1$ gefunden wurde. Da es ein perfektes Matching gibt [BDD+04], muss es auch einen Pfad der Länge $k-1$ geben.

Um den Matching-Graphen aufzubauen, wird ein effizientes Verfahren benötigt, das für jede Komponente alle realisierbaren Matching-Konfigurationen bestimmt.

Sei G_i eine Komponente und $(T_{i,\text{Matched}}, T_{i,\text{Free}})$ eine Matching-Konfiguration. Bestimme mit dem Verfahren aus Kapitel 7 einen Hamiltonkreis in G_i. Damit lässt sich ein perfektes oder fast perfektes Matching M_i' bestimmen. Entferne nun aus G_i alle Kanten, die in G nicht vorhanden sind, sowie alle Knoten aus $T_{i,\text{Matched}}$ und alle Knoten aus $T_{i,\text{Free}}$. Dadurch wird nur eine konstante Anzahl an Knoten frei, da nur eine konstante Anzahl von Knoten und Kanten entfernt wurde. Im restlichen Graphen kann nun durch Auffinden einer konstanten Anzahl von augmentierenden Pfaden ein größtes Matching gefunden werden. Ist

dieses Matching perfekt, so ist die gegebene Konfiguration realisierbar, anderenfalls ist sie nicht realisierbar.

Da es pro Komponente nur konstant viele Matching-Konfigurationen gibt, können die realisierbaren Matching-Konfigurationen in einer Komponte G_i mit $n' = |G_i|$ in $O(n'\alpha(n'))$ Zeit berechnet werden.

Sei $n_i = |G_i|$. Da jede Komponente höchstens 6 Knoten mit anderen Komponenten teilt, gilt: $\sum_{i=1}^{k} n_i \leq n + 6k \leq 7n$. Die Berechnung aller realisierbaren Matching-Konfigurationen für alle Komponenten benötigt also folgenden Aufwand:

$$\sum_{i=1}^{k} n_i\alpha(n_i) \leq \sum_{i=1}^{k} n_i \cdot \max_j \alpha(n_j) \leq 7n \cdot \alpha(n).$$

Demnach können alle realisierbaren Matching-Konfigurationen für alle Komponenten in insgesamt $O(n\alpha(n))$ Zeit bestimmt werden.

Anschließend muss, wenn der Graph ungerade Kardinalität hatte, noch der zuvor hinzugefügte Knoten entfernt werden. Dadurch wird wieder ein Knoten frei. Insgesamt findet dieses Verfahren ein (fast) perfektes Matching in beinahe Linearzeit:

Satz 16. *Sei G ein dreifach zusammenhängender planarer Graph, dessen 4-Blockbaum ein Pfad ist, dann kann ein (fast) perfektes Matching in G in $O(n\alpha(n))$ Zeit berechnet werden.*

Dieses Resultat wird im nächsten Abschnitt verwendet, um ähnlich wie in Kapitel 6 Blätter des 4-Blockbaums abzuschneiden und so Matchings in beliebigen dreifach zusammenhängenden planaren Graphen zu berechnen.

8.5 Der Fall $\ell_4 > 2$

Sei G ein dreifach zusammenhängender planarer Graph und $\ell_4 > 2$. Wie zuvor kann der 4-Blockbaum des Graphen in $O(n\alpha(n))$ Zeit berechnet werden. Ähnlich wie in Abschnitt 6.3 sollen nun Blätter des 4-Blockbaums abgetrennt werden und im abgetrennten Teilgraphen und der Hauptkomponente getrennt Matchings berechnet werden. Anschließend sollen beide Matchings zu einem Matching im ganzen Graphen zusammengefügt werden. Da hier stets einzelne Blätter abgeschnitten werden, ist keine Knotenkontraktion nötig, um den 4-Blockbaum der Hauptkomponente zu verwalten.

Suche zunächst ein beliebiges Tripel zum Abschneiden eines Blattes. Wähle dazu ein beliebiges Blatt des 4-Blockbaums und laufe solange aufwärts im Baum, bis ein Knoten vom Grad mindestens 3 erreicht wird. Die zuletzt benutzte Kante gehört zu einem separierenden Tripel T, das zum Abschneiden eines Blattes verwendet werden kann. Betrachte dazu $G - T$. Dieses zerfällt in zwei Komponenten; Zuerst die Hauptkomponente, sie enthält die Komponentente, die im 4-Blockbaum von G mindestens Grad 3 hat. Die andere Komponente heißt

Zweig. Füge nun T zusammen mit allen Kanten aus G, die in der jeweiligen Komponente in G existieren, sowie den Kanten, die die Knoten in T paarweise miteinander verbinden, zu beiden Komponenten hinzu. Nun kann in beiden Komponenten ein Matching berechnet werden. Für die Hauptkomponente wird hierzu Rekursion verwendet. Für den Zweig gilt in jedem Fall $\ell_4 \leq 2$. Ist die Kardinalität des Zweiges ungerade, so füge einen neuen Knoten v in die Facette ein, die von T begrenzt wird und verbinde ihn mit allen Knoten aus T. Dieser wird nach Berechnung des perfekten Matchings wieder entfernt. Da er zu einem Knoten von T gematcht sein muss, wird ein Knoten aus T frei.

Nun müssen die beiden Matchings aus der Hauptkomponente und dem Zweig zusammengefügt werden. Dabei ist zu beachten, dass die Kanten, die die Knoten von $T = \{u, v, w\}$ miteinander verbinden, im Originalgraphen eventuell nicht vorhanden sind und daher die Abschätzung verschlechtern. Zudem sind Knoten von T eventuell in beiden Komponenten zu unterschiedlichen Knoten gematcht. Die Matchings können aber stets so kombiniert werden, dass höchstens drei weitere Knoten frei werden. Dazu ist eine Fallunterscheidung nötig:

Fall 1: Eine Kante, die zwei Knoten von T verbindet, etwa uv, ist im Matching der Hauptkomponente enthalten.

Fall 1.1: Auch im Matching des Zweiges ist eine solche Kante enthalten.

Ist dieselbe Kante auch im Matching des Zweiges enthalten, so ist diese Kante im schlimmsten Falle gar nicht in G vorhanden. Durch das Entfernen der Kante werden also die Knoten u, v frei. Der Knoten w kann ebenfalls in beiden Komponenten zu unterschiedlichen Knoten gematcht sein. Hatte der Zweig ungerade Kardinalität, so muss w im Zweig zum Hilfsknoten gematcht sein und er kann in der Hauptkomponente gematcht werden, ohne dass weitere Knoten frei werden. Hatte der Zweig gerade Kardinaltität, so kann der Knoten nur in einer Komponente verwendet werden und es wird höchstens ein weiterer Knoten frei. Die hängt davon ab, ob w in der Hauptkomponente überhaupt gematcht war oder nicht.

Im Matching des Zweiges könnte aber auch eine andere Kante des Tripels enthalten, sein als in der Hauptkomponente, etwa vw. Werden beide Kanten uv, vw entfernt, so entstehen dadurch höchstens drei freie Knoten. Hat der Zweig ungerade Kardinalität, so muss der zusätzliche Knoten zu u gematcht sein. Dieser ist dann bereits frei und wird daher nicht erneut gezählt. Hat der Zweig gerade Kardinalität, so ist u zu einem anderen Knoten v' des Zweiges gematcht. Dann kann die Kante uv' mit ins Matching aufgenommen werden und die Anzahl der freien Knoten ist sogar nur zwei.

Fall 1.2: Im Matching des Zweiges ist keine der Kanten uv, vw, wu enthalten.

Entferne die Kante uv aus dem Matching der Hauptkomponente. Diese sind im Zweig gematcht, so dass dadurch keine freien Knoten entstehen. Nur der Knoten w ist eventuell in beiden Komponenten gematcht. Wird eine der beiden möglichen Kanten ins Matching aufgenommen, so wird

dadurch höchstens ein Knoten frei. Besitzt der Zweig ungerade Kardinalität, so entsteht durch Entfernen des Hilfsknotens höchstens ein weiterer freier Knoten. Insgesamt bleiben höchstens zwei Knoten frei.

Fall 2: Das Matching der Hauptkomponenten verwendet keine der Kanten uv, vw, wu.

Fall 2.1: Das Matching des Zweiges verwendet eine der Kanten des Tripels, etwa uv. Entferne dann einfach uv aus dem Matching. Dadurch werden keine Knoten frei, da u, v in der Hauptkomponente bereits gematcht sind oder schon als frei gezählt wurden. Ist w in beiden Komponenten gematcht, so wird eine der beiden möglichen Kanten entfernt. Dadurch entsteht ein freier Knoten. Besitzt der Zweig ungerade Kardinalität, so muss w zum Hilfsknoten gematcht sein, so dass tatsächlich gar kein zusätzlicher Knoten frei wird.

Fall 2.2: Keines der beiden Matchings verwendet eine der Kanten uv, vw und wu. In diesem Fall besteht für jeden der Knoten u, v, w die Wahl zwischen je zwei Kanten, die ins Matching aufgenommen werden können. Es können einfach die Kanten aus dem Matching des Zweiges gewählt werden. Die Kanten des Matchings der Hauptkomponente, die zu u, v, w inzident sind, werden entfernt. Dadurch entstehen höchstens drei zusätzliche freie Knoten. Besitzt der Zweig gerade Kardinalität, so ist nichts weiter zu tun. Hat der Zweig ungerade Kardinalität, so ist einer der Knoten u, v, w, nach Entfernen des Hilfsknotens frei. Sei u dieser Knoten. Ist u im Matching der Hauptkomponente frei, so braucht er nicht erneut gezählt zu werden. Anderenfalls kann u in der Hauptkomponente gematcht werden und die Anzahl der freien Knoten ist sogar nur zwei.

Das Abschneiden eines Zweiges und die getrennte Berechnung von Matchings erzeugt also höchstens drei freie Knoten pro Blatt des 4-Blockbaumes. Der folgende Satz fasst dies zusammen.

Satz 17. *Sei G ein dreifach zusammenhängender planarer Graph, dessen 4-Blockbaum ℓ_4 Blätter besitzt. Dann kann in $O(n\alpha(n))$ Zeit ein Matching der Größe mindestens $(2n - 6\ell_4)/2$ berechnet werden.*

Beweis. Die Schranke für die Kardinalität des Matchings ist äquivalent dazu, dass höchstens $3\ell_4$ Knoten bezüglich M frei sind.

Der 4-Blockbaum von G kann in $O(n\alpha(n))$ Zeit bestimmt werden [KTBC92]. Ist $\ell_4 \leq 2$, so kann das Matching mit Hilfe eines Hamiltonkreises bzw. dem Algorithmus aus dem vorigen Abschnitt direkt bestimmt werden. Gilt $\ell_4 > 2$, so wird im 4-Blockbaum ein Tripel gesucht, an dem ein Zweig so abgeschnitten werden kann, dass der 4-Blockbaum der Hauptkomponente ein Blatt weniger hat als der 4-Blockbaum von G. Wie zuvor gesehen, kann nun an diesem Tripel abgeschnitten werden und in beiden Komponenten getrennt Matchings bestimmt werden. Für die Hauptkomponente H gilt $\ell_4(H) = \ell_4(G) - 1$ und nach Induktionsvoraussetzung existiert dort ein Matching, bei dem höchstens $3\ell_4(H) = 3\ell_4(G) - 3$ Knoten frei sind. Oben wurde gezeigt, dass durch das Zusammenfügen der Matchings der beiden Komponenten höchstens 3 weitere

Knoten frei werden. Daher hat G bezüglich des zusammengefügten Matchings höchstens $3\ell_4$ freie Knoten und für das Matching M gilt $|M| \geq (n - 3\ell_4)/2$.

Die Laufzeit ergibt sich, indem die einzelnen Operationen betrachtet werden. Es werden ℓ_4 Abschneideoperationen durchgeführt. Dabei wird jeweils eine konstante Anzahl von Knoten und Kanten hinzugefügt. Da $\ell_4 \leq (2n - 4)/3$ [BDD+04], ist die Anzahl der Knoten und Kanten nach der vollständigen Zerlegung noch immer in $O(n)$. Die Matchings in den einzelnen Komponenten können dann mit den zuvor angegebenen Verfahren in $O(n\alpha(n))$ gefunden werden. Danach sind wieder ℓ_4 Schritte nötig, um das Matching zusammenzufügen. Jeder dieser Schritte benötigt nur konstante Zeit. Damit ergibt sich ein Gesamtaufwand von $O(n\alpha(n))$. □

In jedem dreifach zusammenhängenden planaren Graphen G existiert ein Matching der Größe mindestens $(2n + 4 - \ell_4)/4$ [BDD+04]. Der hier angegebene Algorithmus berechnet eines der Größe $(n - 3\ell_4)/2 = (2n - 6\ell_4)/2$. Durch Auffinden von zwei augmentierenden Pfaden in $O(n\alpha(n))$ Zeit kann die Größe auf $(2n + 4 - 6\ell_4)/2$ vergrößert werden. Dadurch lassen sich die Schranken besser miteinander vergleichen.

Der Algorithmus lässt pro Blatt zu viele Knoten frei, um die Schranke von $(2n + 4 - \ell_4)/4$ zu erreichen. Ein Problem stellt das Zusammenfügen von Matchings dar. Wäre es möglich, das Matching im Zweig passend zur Vorgabe aus der Hauptkomponente zu wählen, so wären weniger freie Knoten pro Blatt nötig. Insbesondere würde die Existenz eines geeigneten Matchings im Zweig genügen, da sich dieses dann auch mit dem Verfahren aus Abschnitt 8.4 schnell finden ließe, indem die gewünschte Konfiguration als Startknoten für die Suche im Matchinggraphen verwendet wird.

Da die Schranke $(2n + 4 - \ell_4)/4$ scharf ist [BDD+04], ist es nicht möglich, Zweige so abzuschneiden, dass dabei nie ein freies Blatt entsteht. Jedes Vorgehen beim Abschneiden von einzelnen Zweigen kann also zu mindestens einem freien Knoten führen.

Im nächsten Abschnitt wird für trianguläre Graphen gezeigt, dass das Matching im Blatt passend zur Hauptkomponente gewählt werden kann. Dadurch verbessert sich die Schranke auf $(2n + 4 - 2\ell_4)/4$, also einen freien Knoten pro Blatt des 4-Blockbaums.

Gelänge es, zu zeigen, dass jeder dreifach zusammenhängende planare Graph, dessen 4-Blockbaum ein Pfad ist, ein (fast) perfektes Matching besitzt, das eine gegebene Konfiguration am Tripel, das zum Abschneiden verwendet wird, realisiert, so könnte der Algorithmus auch für beliebige dreifach zusammenhängende planare Graphen verbessert werden. Im Algorithmus müsste dann die Suche im Matchinggraphen einfach am der gewünschten Startkonfiguration begonnen werden. Ob ein solches Matching stets existiert ist eine offene Frage dieser Arbeit.

Dennoch genügt dies nicht, um die Schranke von Biedl et al. zu erreichen, da diese nur einen freien Knoten für je zwei Blätter des 4-Blockbaums gestattet. Die Blätter müssten also paarweise auf Kosten jeweils eines freien Blattes abgeschnitten werden. Es ist nicht klar, wie und ob das effizient zu machen ist.

Die Lösung dieser Frage würde für triangulierte Graphen vermutlich direkt die Schranke von Biedl et al. von $(2n+4-\ell_2)/4$ erreichen. Zusammen mit der Klärung der vorigen Frage besteht die Hoffnung, diese Schranke auch für beliebige dreifach zusammenhängende planare Graphen zu erreichen.

8.6 Triangulierte Graphen

Sei G ein triangulierter planarer Graph mit mindestens vier Knoten. Dann ist G dreifach zusammenhängend [KN05], und das Verfahren aus dem vorigen Abschnitt kann direkt eingesetzt werden. Zudem ist jedes separierende Tripel $\{u, v, w\}$ ein separierendes Dreieck, die Kanten uv, vw, wu sind also in G vorhanden.

Bezeichne ℓ_4 wieder die Anzahl der Blätter des 4-Blockbaums. Probleme beim Abschätzen der freien Knoten im vorigen Abschnitt machte die Tatsache, dass das Matching in einem Zweig nicht an die Situation in der Hauptkomponente angepasst war. In triangulierten Graphen ist die Situation einfacher, da die Kanten, die ein separierendes Tripel verbinden, tatsächlich vorhanden sind.

Sei wieder $\ell_4 = 2$, die Vierfachzusammenhangskomponenten bilden also wieder einen Pfad. Beginnt man mit der Berechnung in einer Blattkomponente, so ist in jeder der folgenden Komponenten eine Knotenmenge von bereits zuvor gematchten Knoten gegeben. Außerdem dürfen nur Knoten freigelassen werden, die zum separierenden Tripel, das von der nächsten Komponente separiert, gehören.

Zunächst ist die Anzahl der bereits gematchten Knoten immer zwei oder drei: Die Komponente hat mit ihrer Vorgängerkomponente nur drei Knoten gemeinsam. Wären weniger als zwei Knoten davon bereits gematcht, etwa u, v frei, so kann die Kante uv ins Matching der Vorgängerkomponente aufgenommen werden und so die Anzahl der bereits gematchten Knoten um zwei erhöht werden. Es ist zu beachten, dass die separierenden Tripel, die die Komponente von ihrem Vorgänger und ihrem Nachfolger abtrennen, nicht disjunkt sein müssen. Auf Grund der Planarität können sie aber nicht identisch sein.

Nun soll eine Vorgabe, welche Knoten bereits gematcht sind, fortgesetzt werden, so dass nur Knoten des nächsten Tripels frei sind. Gelingt dies, so kann der Algorithmus aus Abschnitt 8.4 vereinfacht werden. Dann ist nämlich klar, dass jede Konfiguration fortsetzbar ist und es müssen daher nicht alle Möglichkeiten lokal durchprobiert werden. Tatsächlich genügt es, die Existenz einer geeigneten Fortsetzung zu zeigen. Dann kann ein Matching mit einem Hamiltonkreis bestimmt werden. Anschließend werden alle Knoten gelöscht, die bereits gematcht sind oder frei bleiben sollen. Da das nur eine konstante Anzahl an Knoten ist, kann das Matching dann in $O(n\alpha(n))$ Zeit zu einem geeigneten Matching erweitert werden. Dabei bezeichnet n hier die Anzahl der Knoten in der Komponente, nicht die Größe des Gesamtgraphen.

Je nach Kardinalität der Komponente und Anzahl der bereits gematchten Knoten ist ein leicht modifiziertes Vorgehen nötig:

Fall 1: Es sind bereits zwei Knoten u, v gematcht.

Dann besitzt $C - \{u, v\}$ einen Hamiltonkreis [TY94]. Besitzt die Komponente eine ungerade Anzahl von Knoten, so kann der freizulassende Knoten so gewählt werden, dass einer der Knoten des nächsten separierenden Tripels frei bleibt.

Fall 2: Alle drei Knoten u, v, w sind bereits gematcht.

$C - u$ besitzt einen Hamiltonkreis H, der die Kante vw enthält [TY94]. Ist $|C|$ ungerade, so enthält H ein perfektes Matching. Dieses kann so gewählt werden, dass die Kante vw darin enthalten ist.

Ist $|C|$ gerade, so entferne u zusammen mit einem der Knoten des nächsten separierenden Tripels, der nicht zu u, v, w gehört. Dieser Graph besitzt dann einen Hamiltonkreis H, der die Kante vw enthält [TY94]. Wie zuvor enthält dieser Kreis ein geeignetes Matching.

Also lässt sich jede vorgegebene Konfiguration zu einem (fast) perfekten Matching fortsetzen. Besitzt der Zweig ungerade Kardinalität, so bleibt dadurch nur ein Knoten der letzten Komponente frei.

Verwendet man dieses Vorgehen zum Abschneiden eines Zweiges wie im vorherigen Abschnitt, so kann das Matching der Hauptkomponente als Vorgabe für die gematchten Knoten in der ersten Komponente des abgeschnittenen Zweiges interpretiert werden. Dadurch bleibt durch das Abschneiden eines Zweiges höchstens ein zusätzlicher Knoten frei, nämlich genau dann, wenn in der Hauptkomponente alle drei Knoten schon gematcht sind und der Zweig gerade Kardinalität besitzt. Insgesamt ergibt sich folgender Satz:

Satz 18. *Sei G ein triangulierter Graph, dessen 4-Blockbaum ℓ_4 Blätter besitzt. Dann kann ein Matching der Größe mindestens $(2n + 4 - 2\ell_4)/4$ in $O(n\alpha(n))$ Zeit berechnet werden.*

9 Graphen mit gradbeschränkten Blockbäumen

In diesem Kapitel wird ein Verfahren vorgestellt, mit dem es möglich ist, in speziellen Graphen in $o(\sqrt{n}m)$ Zeit *größte* Matchings zu bestimmen. In Kapitel 6 wurde ein Verfahren angegeben, das in 3-regulären Graphen ein Matching der Größe $(3n - 2\ell_2)/6$ in $O(n \log^4 n)$ Zeit findet. In Kapitel 8 wurde gezeigt, wie in dreifach zusammenhängenden planaren Graphen ein Matching der Größe $(2n + 4 - 6\ell_4)/4$ in $O(n\alpha(n))$ Zeit gefunden werden kann. Dabei bezeichnen ℓ_2 und ℓ_4 die Anzahl der Blätter im 2- bzw. 4-Blockbaum. Ist ℓ_2 oder ℓ_4 durch eine Konstante beschränkt, so können mit Hilfe von augmentierenden Pfaden mit dem gleichen asymptotischen Aufwand in der jeweiligen Graphenklasse sogar größte Matchings gefunden werden.

Diese Einschränkung soll nun dahingehend gelockert werden, dass nicht die Anzahl der Blätter durch eine Konstante beschränkt wird, sondern der Knotengrad des 2- bzw. 4-Blockbaums. Jede Zwei- bzw. Vierfach-Zusammenhangskomponente ist folglich zu einer konstanten Anzahl anderer Komponenten inzident. Ähnlich wie in Abschnitt 8.4 ist für die Konstruktion eines Matchings in zwei benachbarten Komponenten nur die Information, welche der *gemeinsamen* Knoten in welcher Komponente gematcht werden, von Belang. Die genaue Struktur des Matchings in den restlichen Komponenten ist nicht wichtig. Das Verfahren verwendet dynamische Programmierung, um verschiedene Matchings in den Komponenten zu einem größten Matching im gesamten Graphen zusammenzusetzen. Die Laufzeit ist exponentiell im Maximalgrad des zugehörigen Blockbaums.

Im Folgenden wird das Verfahren für dreifach zusammenhängende planare Graphen mit gradbeschränktem 4-Blockbaum beschrieben. Anschließend wird gezeigt, welche Änderungen nötig sind, um damit auch größte Matchings in 3-regulären Graphen mit gradbeschränktem 2-Blockbaum zu finden.

Biedl [Bie01] zeigt, dass sich das Problem, größte Matchings in beliebigen Graphen zu finden, in Linearzeit auf das Finden von größten Matchings in 3-regulären Graphen reduzieren lässt. Für größte Matchings in planaren Graphen gibt es ebenfalls eine Linearzeitreduktion auf größte Matchings in triangulierten Graphen mit Maximalgrad 9 [Bie01]. Daher ist es unwahrscheinlich, dass sich die hier angegebenen Verfahren noch verbessern lassen.

9.1 Dreifach zusammenhängende planare Graphen

Sei G ein dreifach zusammenhängender planarer Graph. Zunächst wird wie in Kapitel 8 sein 4-Blockbaum berechnet. Sei k der Maximalgrad des 4-Block-

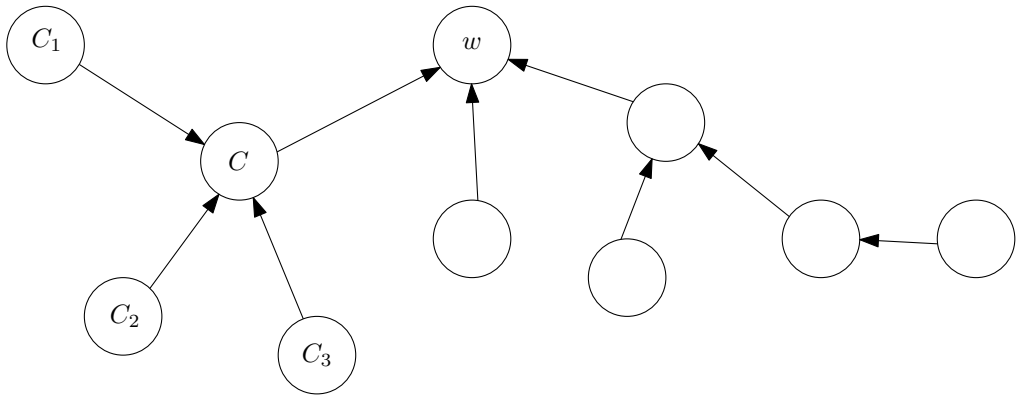


Abbildung 9.1: Blockbaum mit gerichteten Kanten.

baums. In den einzelnen Komponenten werden nach und nach Matchings berechnet. Dabei werden nur Matchings in Knoten berechnet, für die bereits in allen bis auf höchstens einen Nachbarn Matchings berechnet wurden. Eine solche Komponente heißt *zulässige Komponente*. Es wird also in den Blättern begonnen, da diese zu Beginn die einzigen zulässigen Komponenten sind. Die Reihenfolge, in der Matchings in den Komponenten konstruiert werden, kann folgendermaßen bestimmt werden:

Wähle einen beliebigen Knoten w des 4-Blockbaums als Wurzel und richte alle Kanten des 4-Blockbaums zur Wurzel hin aus. Dadurch führen alle Wege zu w. Eine topologische Sortierung dieses gerichteten Graphen liefert dann eine korrekte Reihenfolge zur Bearbeitung der Komponenten. Die zu w gehörige Komponente wird als letzte bearbeitet. Sobald sie bearbeitet wurde, terminiert die Berechnung. Abbildung 9.1 zeigt den Baum, nachdem die Kanten gerichtet wurden. Die Komponente C ist beispielsweise erst dann zulässig, wenn C_1, C_2 und C_3 bereits bearbeitet wurden.

Jede Kante dieses Blockbaums entspricht einem separierenden Tripel, also drei Knoten, die in beiden inzidenten Komponenten enthalten sind. Das Tripel, das der ausgehenden Kante entspricht, heißt *ausgehendes Tripel*. Jeder Knoten erhält einen Zähler für jede Möglichkeit, welche Knoten des ausgehenden Tripels bereits in der Komponente oder einem ihrer Vorgänger gematcht sein können. Da das ausgehende Tripel T aus drei Knoten besteht und jeder davon entweder schon gematcht oder frei sein kann, besitzt jede Komponente $|\mathcal{P}(T)| = 8$ Zähler, für jede Teilmenge von T einen. Dabei beizeichnet $\mathcal{P}(A)$ die Potenzmenge einer Menge A, also die Menge aller Teilmengen von A.

Für jeden Knoten v des Blockbaums bilden alle Knoten, von denen ein Pfad entlang der gerichteten Kanten zu v führt, einen Baum. Dieser Baum enthält alle Vorgänger von v. Sei C eine solche Komponente. Dann bezeichne G_C den Teilgraphen von G, der die Knoten aller Vorgängerkomponenten von C und die Knoten von C enthält. In Abbildung 9.1 sind C_1, C_2 und C_3 die Vorgänger von C. Der Graph G_C enthält in diesem Fall alle Knoten, die in C, C_1, C_2 oder C_3 enthalten sind.

Sei T das ausgehende separierende Tripel von C. Für $F \subseteq T$ beschreibt der Zähler $c_C(F)$ der Komponente C die Anzahl an Knoten von $G_C \setminus F$, die bezüglich einem größten Matching in $G_C \setminus F$ frei bleiben müssen. Im Folgenden

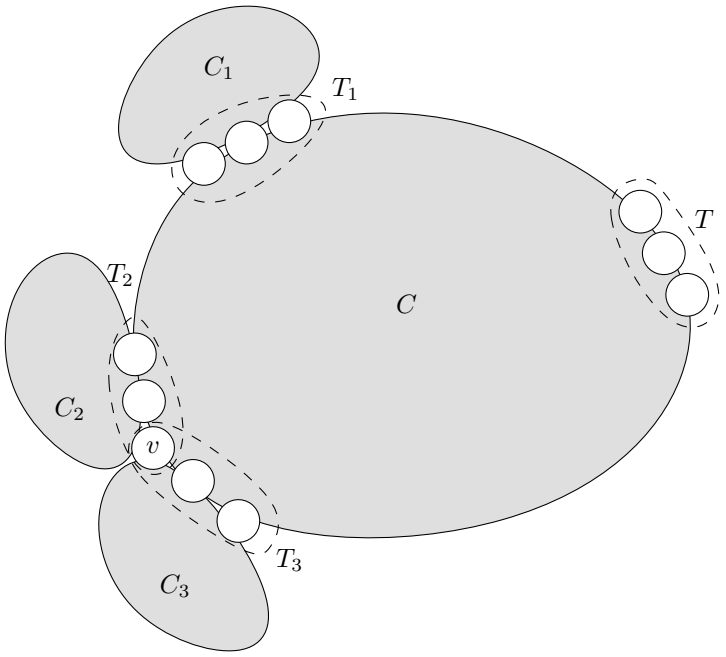


Abbildung 9.2: Berechnung der Zähler einer Komponente C.

wird gezeigt, wie diese Werte mittels dynamischer Programmierung berechnet werden können. Dazu werden zunächst sämtliche Zähler mit ∞ initialisiert.

Für eine Blattkomponente C können die Zähler leicht mit dem Verfahren aus Abschnitt 7.4 in $O(n\alpha(n))$ Zeit berechnet werden. Zusätzlich kann zu jedem Zähler das entsprechende Matching mit abgespeichert werden.

Sei C nun eine beliebige zulässige Komponente, $C_1, \ldots, C_r$ seine direkten Vorgänger und T das ausgehende Tripel, also das Tripel, das C von der nächsten noch unbearbeiteten Komponente separiert. Da jede Komponente höchstens zu k Komponenten adjazent ist, gilt $r \leq k - 1$. Außerdem bezeichne T_i das Tripel, das C_i von C separiert. Ein Beispiel für diese Situation ist in Abbildung 9.2 zu sehen. Dort ist der Knoten v sowohl in T_2, als auch in T_3 enthalten.

Sei $F \subseteq T$. Betrachtet man ein beliebiges größtes Matching M in $G_C - F$ und schränkt dieses auf C_i ein, so sind eventuell einige einige Knoten F_i von T_i frei. Der Zähler $c_{C_i}(F_i)$ besagt, wieviele Knoten bei einem größten Matching in $G_{C_i} \setminus F_i$ frei bleiben. Zusätzlich ist die Information abgelegt, wie dieses Matching in C_i aussieht. Da nicht von vornherein klar ist, welche F_i für die Vorgängerkomponenten zu wählen ist, wird für jede Vorgängerkomponente jede der acht Möglichkeiten mitgeführt.

Eine Möglichkeit, wie die Matchings zusammengesetzt werden können, kann durch ein Tupel $(F_1, \ldots, F_r) \in \mathcal{P}(T_1) \times \cdots \times \mathcal{P}(T_r)$ beschrieben werden, eine sogenannte *Konfiguration*. Dabei sei $F_i \subseteq T_i$ die Menge der freigelassenen Knoten in C_i. Die Knoten in $S_i := T_i \setminus F_i$ sind also für die Komponente C_i reserviert, unabhängig davon, ob sie dort verwendet werden oder nicht. Da $|\mathcal{P}(T_1) \times \cdots \times \mathcal{P}(T_r)| = 8^r \leq 8^k = O(1)$ können alle Kombinationen effizient durchprobiert werden.

Eine einzelne Konfiguration $(F_1, \ldots, F_r)$ wird folgendermaßen geprüft:

- Zunächst muss die Konfiguration so beschaffen sein, dass sich die Matchings der Vorgängerkomponenten zusammensetzen lassen: Die Mengen der für die Vorgängerkomponenten reservierten Knoten S_i müssen daher paarweise disjunkt sein und dürfen auch keinen Knoten aus F enthalten. Ist dies der Fall, so heißt die Konfiguration $(F, F_1, \ldots, F_r)$ *zulässig*. Ist dies nicht der Fall, so sind die Kosten für diese Konfiguration:

$$\text{cost}(F, F_1, \ldots, F_r) = \infty.$$

- Bestimme nun ein größtes Matching $M(F, F_1, \ldots, F_r)$ in $C \setminus (\bigcup_{i=1}^{r} S_i \cup F)$, das keine Kanten verwendet, die nicht in G vorhanden waren. Bezeichnet n' die Anzahl der Knoten von C, so ist das, da nur eine konstante Anzahl an Knoten und Kanten nicht verwendet werden darf, nach Lemma 13 in $O(n'\alpha(n'))$ Zeit möglich, siehe Abschnitt 7.4.
- $f(F, F_1, \ldots, F_r)$ bezeichne die Anzahl der in $C \setminus (\bigcup_{i=1}^{r} S_i \cup F)$ bezüglich $M(F, F_1, \ldots, F_r)$ freien Knoten.
- Die Kosten der Konfiguration $(F, F_1, \ldots, F_r)$ sind dann:

$$\text{cost}(F, F_1, \ldots, F_r) = \sum_{i=1}^{r} c_{C_i}(F_i) + f(F, F_1, \ldots, F_r).$$

  Das ist genau die Anzahl der Knoten, die in den Graphen G_{C_i} frei bleiben muss, um dort ein größtes Matching zu realisieren, das die Knoten in F freilässt und dessen Einschränkung auf G_{C_i} die Knoten F_i freilässt. Dazu kommt noch die Anzahl der Knoten, die auch in Komponente C nicht gematcht werden können, wenn die Knoten in F frei bleiben sollen, nämlich $f(F, F_1, \ldots, F_r)$.

Die Kosten einer einzelnen Konfiguration können also in $O(n'\alpha(n'))$ Zeit berechnet werden. Der Zähler $c_C(F)$ wird auf den Wert

$$c_C(F) := \min\{\text{cost}(F, F_1, \ldots, F_r) \mid (F, F_1, \ldots, F_r) \text{ Konfiguration von } C\}$$

gesetzt. Dieser Wert entspricht der Anzahl an Knoten, die bezüglich eines größten Matchings in G_C, das die Knoten in F frei lässt, zusätzlich frei bleiben müssen. Da nur eine konstante Anzahl an Kombinationen durchprobiert werden muss und jede davon $O(n'\alpha(n'))$ Zeit benötigt, kann $c_C(F)$ in $O(n'\alpha(n'))$ Zeit berechnet werden. Außerdem kann die für diesen Wert zugehörige Konfiguration $(F_1, \ldots, F_r)$ und das zugehörige Matching $M(F, F_1, \ldots, F_r)$ mit abgespeichert werden.

Dieser Vorgang wird für jedes $F \subseteq T$ durchgeführt. Da es nur acht Möglichkeiten gibt, benötigt das Berechnen aller Zähler $c_C(F)$ nur $O(n\alpha(n))$ Zeit. Zu jedem Zähler wird dabei auch die beste zugehörige Konfiguration sowie das zugehörige Matching mit abgelegt.

So werden nach und nach sämtliche Komponenten des Graphen verarbeitet, bis nur noch eine Komponente übrig ist, die Komponente die zum Knoten w im 4-Blockbaum gehört. Ab jetzt bezeichne C diese Komponente, $C_1, \ldots, C_r$ seien

alle inzidenten Komponenten und T_i sei das Tripel, das C_i von C separiert. Bei der letzten Komponente gibt es keinen Nachfolger mehr. Es werden nur noch alle Konfigurationen der direkten Vorgänger durchgegangen, und es wird geprüft, welche sich mit den wenigsten freien Knoten zusammensetzen lässt. Es werden also alle Konfigurationen $(F_1, \ldots, F_r) \in P(T_1) \times \cdots \times P(T_r)$ untersucht. Dazu wird jeweils wie zuvor ein Matching $M(F_1, \ldots, F_r)$ in C bestimmt, das keine Knoten aus $S_i := T_i \setminus F_i$ verwendet und auch keine Kanten, die in G nicht vorhanden sind. Wie zuvor bezeichne $f(F_1, \ldots, F_r)$ die Anzahl der in $G_C \setminus \bigcup_{i=1}^{r} S_i$ bezüglich $M(F_1, \ldots, F_r)$ freien Knoten. Die Kosten einer Konfiguration $F_1, \ldots, F_r$ werden wie zuvor berechnet:

$$\text{cost}(F_1, \ldots, F_r) = \sum_{i=1}^{r} c_{C_i}(F_i) + f(F_1, \ldots, F_r).$$

Die Anzahl der freien Knoten für ein größtes Matching in G ist dann:

$$\text{cost} = \min\{\text{cost}(F_1, \ldots, F_r) : (F_1, \ldots, F_r) \in P(T_1) \times \cdots \times P(T_r)\}.$$

Wie zuvor gibt es nur eine konstante Anzahl an Kombinationen, die alle in $O(n'\alpha(n'))$ Zeit, $n' = |C|$ getestet werden können. Zusätzlich zu den kleinsten Kosten wird auch eine beste Konfiguration $F_1, \ldots, F_r$ ermittelt, sowie das zugehörige Matching $M(F_1, \ldots, F_r)$.

Ausgehend davon kann nun duch Rückverfolgung ein Matching angegeben werden, das auch genau die angegeben Anzahl an freien Knoten besitzt:

- Beginne mit dem Matching $M(F_1, \ldots, F_r)$.
- Erweitere dieses rekursiv um das Matching, das in $G_{C_i} \setminus F_i$ nur $c_{C_i}(F_i)$ Knoten frei lässt.

Dabei wird jede Komponente nur einmal besucht, und da es nur $O(n)$ Komponenten gibt, benötigt die Konstruktion des Matchings, nachdem alle Zähler und Konfigurationen berechnet sind, noch $O(n)$ Zeit.

Insgesamt ergibt das folgenden Satz:

Satz 19. *Sei G ein dreifach zusammenhängender planar Graph mit n Knoten. Der 4-Blockbaum von G habe beschränkten Maximalgrad. Dann kann ein größtes Matching in G in $O(n\alpha(n))$ Zeit berechnet werden.*

In einem triangulierten planaren Graphen, dessen 4-Blockbaum Maximalgrad 3 hat, existiert stets ein Hamiltonkreis [JY02] und daher auch ein (fast) perfektes Matching. Ein solches kann mit dem angegeben Verfahren in $O(n\alpha(n))$ Zeit gefunden werden.

Ein ähnliches Verfahren lässt sich auch für 3-reguläre Graphen, deren 2-Blockbaum konstanten Maximalgrad hat, verwenden. Die dafür nötigen Modifikationen werden im nächsten Abschnitt gezeigt.

9.2 3-reguläre Graphen

Sei G ein 3-regulärer Graph, dessen 2-Blockbaum Maximalgrad k hat. Wie in Kapitel 6 kann der 2-Blockbaum in $O(n)$ Zeit berechnet werden. Anschließend wird wie im vorigen Abschnitt ein Wurzelknoten w gewählt und alle Kanten in seine Richtung gerichtet.

Anders als zuvor ist jeder Knoten in genau einer Komponente enthalten. Verbindet eine Brücke v_1v_2 zwei Komponenten C_1, C_2, so kann ein Matching in beiden Einzelkomponenten stets zusammensetzt werden. Sind v_1, v_2 in beiden Komponenten frei, so kann die Brücke zum Matching hinzugenommen werden. Dies muss beim Zusammensetzen der Matchings berücksichtigt werden.

Zunächst wird angenommen, dass jeder Knoten von G zu höchstens einer Brücke inzident ist. Diese Annahme kann später fallen gelassen werden, vereinfacht aber zunächst die Formulierung des Algorithmus.

Die Reihenfolge der Bearbeitung der einzelnen Komponenten ergibt sich wie zuvor. Sei C eine beliebige Komponente, $C_1, \ldots, C_r, r \leq k$ seien die Vorgägner von C. Der Knoten $b_i \in C_i$ sei der Knoten, der zu der Brücke inzident ist, die C_i und C verbindet. Ist C nicht die letzte Komponente, so besitzt sie einen Nachfolger, sei $b \in C$ der Knoten, der zu der Brücke inzident ist, die mit dem Nachfolger verbindet. Die Fortsetzung eines Matchings im Graphen G_C hängt offenbar nur davon ab, ob der Knoten b in C gematcht ist oder nicht. Daher gibt es für jede Komponente C nur zwei Zähler, einen für jede Teilmenge von $\{b\}$. Diese werden nun wie zuvor nach und nach berechnet. Dabei soll nach der Berechnung $c_C(\emptyset)$ die Anzahl der freien Knoten bezüglich eines größtmöglichen Matchings in G_C sein, während $c_C(\{b\})$ die Anzahl der freien Knoten bezüglich eines größten Matchings in $G_C \setminus \{b\}$ enthalten soll.

Durch Entfernen aller Brücken entstehen die Zweifach-Zusammenhangskomponenten. Jede dieser Komponenten enthält nach Entfernen der Brücken eine konstante Anzahl von Knoten mit Grad 2. Diese Knotenmenge sei S. Sie können wie in Abschnitt 6.2 entfernt werden und ihre adjazenten Knoten direkt durch eine Brücke verbunden werden. Dadurch entsteht aus jeder Komponente C ein 3-regulärer zweifach zusammenhängender Graph. In diesem kann dann schnell ein perfektes Matching M bestimmt werden. Die dazu benötigte Zeit sei $T_{\text{Match}}(n')$. Dabei bezeichne n' die Anzahl der Knoten der Komponente.

Anschließend werden die zuvor entfernten Knoten wieder eingefügt. Ausgehend von M kann mit augmentierenden Pfaden für jedes $F \subseteq S$ ein größtes Matching in $G - F$ gefunden werden. Die dafür benötigte Zeit ist in $O(n\alpha(n))$, da bezüglich M nur eine konstante Anzahl von Knoten frei ist.

Anschließend kann wie zuvor von den Blättern ausgehend mittels dynamischer Programmierung ein größtes Matching gefunden werden. Dabei ist zu beachten, dass wenn beide zu einer Brücke inzidenten Knoten frei sind, diese Brücke mit ins Matching aufgenommen werden kann. Da jeder Knoten zu höchstens einer Brücke inzident ist, kann es dabei nicht dazu kommen, dass ein Knoten in zwei benachbarte Komponenten gematcht wird.

Der Gesamtaufwand ist damit in $O(T_{\text{Match}}(n) + n\alpha(n))$. Ist G planar, so sind auch die Komponenten nach Entfernen der Grad-2-Knoten noch planar. In

diesem Fall ist $T_{\text{Match}}(n) = O(n)$. Ist G hingegen nicht planar, so ist $T_{\text{Match}}(n) = O(n \log^4 n)$ [BBDL01].

Es ist einfach, auch Knoten zuzulassen, die zu mehr als einer Brücke inzident sind. Sei v also ein Knoten, der zu mindestens zwei Brücken inzident ist. Da G 3-regulär ist, ist damit *jede* der drei zu v inzidenten Kanten eine Brücke und die Zweifachzusammenhangskomponente von v enthält nur v. Dieser Fall kann leicht geprüft werden. Beim Zusammensetzen der einzelnen Matchings muss nur darauf geachtet werden, dass nicht mehr als eine der inzidenten Brücken in das Matching aufgenommen werden kann. Der folgende Satz fasst die Ergebnisse dieses Kapitels zusammen:

Satz 20. *Sei G ein 3-regulärer Graph mit gradbeschränktem 2-Blockbaum. Ein größtes Matching in G kann in $O(n \log^4 n)$ Zeit gefunden werden. Ist G zusätzlich planar, so kann ein größtes Matching in $O(n\alpha(n))$ Zeit gefunden werden.*

10 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Reihe von Algorithmen zur schnellen Berechnung von Matchings entwickelt. Neben einer Zusammenfassung der wichtigsten Resultate diskutiert dieses Kapitel dabei entstandene beziehungsweise noch nicht beantwortete Fragen. Diese zielen darauf ab, Ergebnisse dieser Arbeit weiter zu verbessern oder zu verallgemeinern.

10.1 Zusammenfassung

Das Hauptanliegen dieser Arbeit war, die Frage zu beantworten, ob Matchings einer gewissen Größe, von denen man weiß, dass sie stets existieren, schneller gefunden werden können als durch Berechnung eines größten Matchings. Die Arbeit hat sich mit einer Reihe von speziellen Graphenklassen beschäftigt, darunter vor allem Graphen mit kleinem Maximalgrad sowie dreifach zusammenhängende planare Graphen. Für diese Graphen wurden Algorithmen entwickelt, mit denen Matchings mit garantierter Mindestgröße in der Tat schneller gefunden werden können als durch Berechnung eines größten Matchings. Dabei werden die meisten der Schranken von Biedl et al. [BDD+04] erreicht. Diese sind für die jeweilige Graphenklasse scharf, so dass kein Algorithmus bessere Schranken auf den jeweiligen Familien garantieren kann. Tabelle 10.1 (eine Wiederholung von Tabelle 1.2 aus Kapitel 1) bietet einen Überblick über die Ergebnisse dieser Diplomarbeit.

Die wichtigsten Ideen und Algorithmen sollen hier noch einmal kurz vorgestellt werden. Eine sehr einfache und trotzdem relativ breit anwendbare Technik ist das Auffinden von Spannbäumen mit beschränktem Maximalgrad. In diesen Bäumen kann dann mit dem Algorithmus aus Kapitel 3 ein größtes Matching gefunden werden. Bereits mit dieser relativ einfachen Konstruktion konnten einige Schranken erreicht werden.

Die Hauptresultate dieser Arbeit sind die Matchingalgorithmen für 3-reguläre Graphen und für dreifach zusammenhängende planare Graphen. Diese Algorithmen basieren auf drei aufeinander aufbauenden Techniken. Die erste wichtige Grundtechnik ist die lokale Betrachtung von Graphen. Lokal erfüllen die Graphen stärkere Eigenschaften, beispielsweise Zweifachzusammenhang. Diese stärkeren Eigenschaften helfen dabei, lokal gute Teilmatchings (etwa in den Zweifachzusammenhangskomponenten) zu finden. Dazu wurde auf bereits bekannte Verfahren zurückgegriffen, die diese Eigenschaften ausnutzen. Der zweite wichtige Aspekt ist der, dass sich der 2- bzw. 4-Blockbaum nutzen lässt, um

Graphenklasse	Schranke für Matchinggröße		Laufzeit
	Schranke 1	Schranke 2	$O(\cdot)$
3-regulär	$(4n-1)/9$	$(3n-2\ell_2)/6$	$n \log^4 n$
Maximalgrad 3	$(n-1)/3$	$(3n-n_2-2\ell_2)/6$	$n \mid n \log^4 n$
$\kappa(G) \geq 3$, planar	$(n+4)/3$	$(2n+4-6\ell_4)/4$	$n\,\alpha(n)$
trianguliert, planar	$(2n+4-2\ell_4)/4$		$n\,\alpha(n)$
Maxgrad k	$(n-1)/k$		n
Maxgrad k	$(m-1)/(2k-2)$		n
Maxgrad k, $\kappa(G) \geq 2$	$2(n-1)/(k+3)$		$n \log^4 n$
Maxgrad k, $\kappa(G) \geq 2$, planar	$2(n-1)/(k+3)$		$n \log^2 n$
Graphen mit gradbeschränktem 2- bzw. 4-Blockbaum			
3-regulär	größtes Matching		$n \log^4 n$
3-regulär, planar	größtes Matching		$n\,\alpha(n)$
$\kappa(G) \geq 3$, planar	größtes Matching		$n\,\alpha(n)$

Tabelle 10.1: Übersicht über die Resultate dieser Arbeit. Dabei bezeichnet κ den Knotenzusammenhang, ℓ_2 die Anzahl der Blätter des 2-Blockbaums und ℓ_4 die Anzahl der Blätter des 4-Blockbaums. Sind zwei durch | getrennte Laufzeiten angeben, so bezieht sich die linke Angabe auf den Algorithmus zum Erreichen von Schranke 1, die rechte Angabe auf Schranke 2.

diese lokalen Matchings geschickt zusammenzusetzen. Dazu war es zunächst nötig, Graphen zu behandeln, deren 2- bzw. 4-Blockbaum ein Pfad ist. Dies ist die zweite wichtige Grundtechnik. Für 3-reguläre Graphen, deren 2-Blockbaum ein Pfad ist, existierte bereits ein schneller Algorithmus zur Berechnung von perfekten Matchings. Ein wesentlicher Beitrag dieser Arbeit besteht darin einen ebensolchen Algorithmus für dreifach zusammenhängende planare Graphen angegeben zu haben, deren 4-Blockbaum ein Pfad ist. Aufbauend auf diesen Algorithmen konnten dann mittels Abschneiden von Blattkomponenten auch Graphen behandelt werden, deren Blockbaum kein Pfad ist. Dieses Abschneiden von Blattkomponenten ist die dritte wichtige Grundtechnik.

Mit diesen Techniken wurde ein Algorithmus angegeben, der in einem 3-regulären Graphen ein Matching der Größe $(3n-2\ell_2)/6$ in $O(n \log^4 n)$ Zeit berechnet. In dreifach zusammenhängenden planaren Graphen kann ein Matching der Größe $(2n+4-6\ell_2)/4$ in $O(n\alpha(n))$ Zeit berechnet werden.

Für 3-reguläre Graphen, sowie dreifach zusammenhängende planare Graphen stellte sich heraus, dass eine Gradbeschränkung des 2- bzw. 4-Blockbaums dazu führt, dass größte Matchings darin in $o(\sqrt{n}m)$ Zeit berechnet werden können. Diese Ergebnisse sind insofern bemerkenswert, als Biedl [Bie01] eine Linearzeitreduktion des Matchingproblems in allgemeinen Graphen auf Matchings in 3-regulären Graphen angibt, sowie eine Linearzeitreduktion von Matchings in planaren Graphen auf Matchings in triangulierten planaren Graphen mit Maximalgrad 9. Dies klärt in gewisser Weise die Frage, welche Strukturen beim Berechnen von Matchings problematisch sind: Zweifach- bzw. Vierfachzusammenhangskomponenten mit vielen Nachbarkomponenten.

10.2 Ausblick

Dieser Abschnitt stellt einige Ansatzpunkte für weitere Untersuchungen vor. Es gibt noch eine Reihe von offenen Fragen:

Kann die Laufzeit um ein Matching der Größe $(3n-2\ell_2)/6$ in einem 3-regulären Graphen zu finden im planaren Fall von $O(n \log^4 n)$ auf $O(n\alpha(n))$ verbessert werden? Zur Beantwortung dieser Frage müsste untersucht werden, ob sich die Blätter des Blockbaums, die abgeschnitten werden, stets so wählen lassen, dass der Hilfsknoten *planar* eingefügt werden kann, siehe Abschnitt 6.4.

In dreifach zusammenhängenden planaren Graphen kann ein Matching der Größe $(2n+4-6\ell_4)/4$ in $O(n\alpha(n))$ Zeit berechnet werden. Ist es möglich auch ein Matching der Größe $(2n+4-\ell_4)/4$ in $O(n\alpha(n))$ Zeit zu finden?

Ein erster Schritt zur Lösung dieser Frage wäre der Nachweis der *Existenz* geeigneter zur Hauptkomponente passender Matchings in dreifach zusammenhängenden planaren Graphen, deren 4-Blockbaum ein Pfad ist. Bereits dadurch ließe sich die Schranke auf $(2n+4-2\ell_4)/4$ verbessern, siehe Abschnitt 8.5.

In einem nächsten Schnitt müsste dann entweder gezeigt werden, dass sich Blätter des 4-Blockbaums paarweise auf Kosten höchstens eines freien Knotens abschneiden lassen, oder dass wenigstens im Schnitt pro Zweig nur ein halber Knoten frei wird. Unabhängig vom vorigen Schritt könnte dies zunächst auch für triangulierte planare Graphen untersucht werden, so dass zumindest für diese die Schranke von Biedl et al. [BDD+04] erreicht würde.

Außerdem haben Nishizeki und Baybars [NB79] noch einige weitere Schranken für die Größe von Matchings in planaren Graphen angegeben. Diese verwenden neben dem Knotenzusammenhang auch den *Minimal*grad um die Existenz großer Matchings nachzuweisen. Ist es möglich, analog zu den Ergebnissen dieser Arbeit auch schnelle Algorithmen anzugeben, die diese Schranken erreichen? In dieser Arbeit wurden insbesondere Eigenschaften wie Maximalgrad, Planarität und Zusammenhang ausgenutzt. Um die Schranken von Nishizeki und Baybars zu erreichen müsste ein Weg gefunden werden Minimalgrade auszunutzen, um große Matchings zu konstruieren.

Literaturverzeichnis

[Alg07] Algorithmic Solutions: *The LEDA User Manual (Version 5.2)*, 2007.

[Bar66] Barnette, David: *Trees in polyhedral graphs.* Canadian J. Mathematics, 18:731–736, 1966.

[BBDL01] Biedl, Therese, Prosenjit Bose, Erik Demaine und Anna Lubiw: *Efficient algorithms for Petersen's theorem.* J. Algorithms, 38:110–134, 2001.

[BDD+04] Biedl, Therese, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer und Stephen G. Kobourov: *Tight Bounds on Maximal and Maximum Matchings.* Discrete Math., 285(1–3):7–15, 2004.

[Ber57] Berge, Claude: *Two theorems in graph theory.* Proc. Natl. Acad. Sci., 43(9):842–844, 1957.

[Bie01] Biedl, Therese: *Linear reductions of maximum matching.* In: *Proc. 12th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'01)*, Seiten 825–826, 2001.

[CLRS01] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introduction to algorithms.* MIT Press, Cambridge, MA, USA, 2. Auflage, 2001.

[CN89] Chiba, Norishige und Takao Nishizeki: *The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs.* J. Algorithms, 10:187–211, 1989.

[CS97] Czumaj, Artur und Willy-Bernhard Strothmann: *Bounded Degree Spanning Trees.* In: *Proc. 5th Annu. European Sympos. Algorithms (ESA'97)*, Band 1284 der Reihe *Lect. Notes Comut. Sci.*, Seiten 104–117. Springer-Verlag, 1997.

[Edm65] Edmonds, Jack: *Paths, trees and flowers.* Canadian Journal of Mathematics, 17:449–467, 1965.

[FR92] Fürer, Martin und Blaji Raghavachari: *Approximating the minimum degree spanning tree to within one from the optimal degree.* In: *Proc. 3rd Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'92)*, Seiten 317–324, 1992.

[GJT76] Garey, Michael R., David S. Johnson und Robert E. Tarjan: *The planar Hamiltonian circuit problem is NP-complete.* SIAM J. Comput., 5:704–714, 1976.

[GKT99] Gabow, Harold N., Haim Kaplan und Robert E. Tarjan: *Unique maximum matching algorithms.* In: *Proc. 31st Annu. ACM Sympos. Theory Comput. (STOC'99)*, Seiten 70–78, 1999.

[HdLT01] Holm, Jacob, Kristian de Lichtenberg und Mikkel Thorup: *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity.* J. ACM, 48(4):723–760, 2001.

[HK73] Hopcroft, John E. und Richard M. Karp: *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs.* SIAM J. Comput., 2:225–231, 1973.

[HT74] Hopcroft, John E. und Robert E. Tarjan: *Efficient Planarity Testing.* J. ACM, 21(4):549–568, 1974.

[JY02] Jackson, Bill und Xingxing Yu: *Hamilton cycles in plane triangulations.* J. Graph Theory, 41(2):138–150, 2002.

[Kan97] Kant, Goos: *A more compact visibility representation.* Intern. J. Comput. Geom. Appl., 7(3):197–210, 1997.

[KN05] Krumke, Sven Oliver und Hartmut Noltenmeier: *Graphentheoretische Konzepte und Algorithmen.* Teubner, Wiesbaden, 2005.

[Kőn16] Kőnig, Dénes: *Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre.* Math. Ann., 77:453–465, 1916.

[Kőn36] Kőnig, Dénes: *Theorie der endlichen und unendlichen Graphen.* Akademische Verlagsgesellschaft, Leipzig, 1936.

[KTBC92] Kanevsky, Arkady, Roberto Tamassia, Giuseppe Di Battista und Jianer Chen: *On-line maintenance of the four-connected components of a graph.* In: *Proc. 33th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS'92)*, Seiten 793–801, 1992.

[MV80] Micali, Silvio und 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)*, Seiten 17–27, 1980.

[NB79] Nishizeki, Takao und Ilker Baybars: *Lower bounds on the cardinality of the maximum matchings of planar graphs.* Discrete Math., 28(3):255–267, 1979.

[Pet91] Petersen, Julius: *Die Theorie der regulären Graphs.* Acta Mathematica, 15:193–220, 1891.

[Rau94] Rauch, Monika: *Improved data structures for fully dynamic biconnectivity.* In: *Proc. 26th Annu. ACM Sympos. Theory Comput. (STOC'94)*, Seiten 686–695, 1994.

[Sch99] Schrijver, Alexander: *Bipartite edge coloring in $O(\Delta m)$ time.* SIAM J. Comput., 28:841–846, 1999.

[Str97] Strothmann, Willy-Bernhard: *Bounded Degree Spanning Trees.* Doktorarbeit, Heinz-Nixdorf-Institut, Universität Paderborn, 1997.

[Tar72] Tarjan, Robert E.: *Depth first search and linear graph algorithms.* SIAM J. Comput., 2:146–160, 1972.

[Tar83] Tarjan, Robert E.: *Data structures and network algorithms.* SIAM, Philadelphia, 1983.

[Tho83] Thomassen, Carsten: *A theorem on paths in planar graphs.* J. Graph Theory, 7:169–176, 1983.

[Tut47] Tutte, William T.: *The factorization of linear graphs.* J. Lond. Math. Soc., 22:107–111, 1947.

[Tut56] Tutte, William T.: *A theorem on planar graphs.* Trans. Amer. Math. Soc., 82:99–116, 1956.

[Tut77] Tutte, William T.: *Bridges and Hamiltonian circuits in planar graphs.* Aequations Math., 15:1–33, 1977.

[Tut84] Tutte, William T.: *Graph Theory*, Band 21 der Reihe *Encycl. Math. Appl.* Addison-Wesley, 1984.

[TY94] Thomas, Robin und Xingxing Yu: *4-connected projective-planar graphs are Hamiltonian.* J. Combinat. Theory Ser. B, 1:114–132, 1994.

[Yan81] Yannakakis, Mihalis: *Edge-Deletion Problems.* SIAM J. Comput., 10(2):297–309, 1981.