Institut für Theoretische Informatik
Lehrstuhl für Algorithmik I

Simultaneous Visualization of Clusterings

Diplomarbeit
von

Jan Christoph Athenstädt

an der Fakultät für Informatik

Gutachter:	Prof. Dr. Dorothea Wagner Prof. Dr. Peter Sanders
Betreuende Mitarbeiter:	Dr. Martin Nöllenburg Dipl.-Inform./Dipl.-Math. Tanja Hartmann

Bearbeitungszeit: 30. November 2012 – 30. Mai 2013

KIT – Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

www.kit.edu

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Karlsruhe, den 30. Mai 2013

(Jan Christoph Athenstädt)

Deutsche Zusammenfassung

Es gibt bereits eine Reihe von Publikationen über die Visualisierung von hierarchischen Clusterungen sowie von überlappenden Teilmengen im Allgemeinen. Allerdings existieren bisher kaum Ansätze, zwei verschiedene Clusterungen simultan zu visualisieren. In dieser Diplomarbeit legen wir die theoretischen Grundlagen für die gleichzeitige Einbettung von zwei oder mehr Clusterungen.

Nach einer Einleitung stellen wir im 2. Kapitel wichtige Konzepte aus den Bereichen Graphen und Hypergraphen vor, welche die Grundlage für unsere Arbeit bilden. Im darauffolgenden Kapitel führen wir Methoden ein, die es uns ermöglichen, Zusammenhänge zwischen Clusterungen zu modellieren.

Im Hauptteil der Arbeit, dem 4. Kapitel, entwickeln wir eine Klassen-Hierarchie, die es uns erlaubt, Familien von Clusterungen in Abhängigkeit von ihrer Einbettbarkeit in die Ebene zu charakterisieren. Für alle bis auf eine Klasse können wir Beispiele angeben, die zeigen, dass die Klassen der Hierarchie echte Teilmengen voneinander sind. Außerdem zeigen wir, wie sich diese Klassen zu bekannten kombinatorischen Problemen verhalten. Für einige Klassen zeigt sich dadurch die Komplexität des Entscheidungsproblems, ob eine Instanz zu dieser Klasse gehört oder nicht.

Für eine der Hauptklassen der Hierarchie, die “Strong Embeddability”, entwickeln wir einen eigenen Beweis um $\mathcal{NP}$-Vollständigkeit zu zeigen. Zur Vorbereitung des Beweises entwickeln wir eine kombinatorische Beschreibung von Einbettungen zweier Clusterungen und definieren eindeutige kombinatorische Einbettungen analog zu planaren Graphen.

Im 5. Kapitel implementieren wir ein ganzzahliges lineares Programm, das es uns erlaubt, optimale Einbettungen für zwei weitere $\mathcal{NP}$-vollständige Klassen zu finden. Außerdem stellen wir einen heuristischen Ansatz vor, der es auch für größere Instanzen erlaubt, gute Lösungen zu finden.

Abschließend führen wir im 6. Kapitel einige Experimente an zufällig generierten Instanzen hinsichtlich der Einbettbarkeit von Clusterungen und der Qualität der Heuristik durch. Zudem wenden wir unsere Methoden im Rahmen einer Fallstudie an zwei Beispielen an, welche auf Echtweltdaten basieren.

Wir beenden die Arbeit mit einem Fazit und Ideen für weiterführende Forschung zum Thema.

Abstract

While there are a number of approaches for the visualization of hierarchical clusterings, as well as subsets in general, there exist only few attempts to visualize different clusterings simultaneously in the same drawing. In this thesis we lay the theoretical foundations for the simultaneous visualization of two or more clusterings. We establish a class-hierarchy that allows us to characterize families of clusterings depending on their embeddability in the plane and show how these classes relate to known combinatorial problems. For some of the classes it turns out to be an $\mathcal{NP}$-complete problem to decide whether an instance belongs to the class or not. We develop and implement an integer linear program that allows us to find optimal embeddings for two classes and additionally provide a simple and fast heuristic that allows us to find a good, yet not always optimal solution. We finally experimentally evaluate our methods on randomly generated instances regarding the embeddability of clusterings and the quality of the heuristic. We conclude this work with a case study in which we apply our methods to two examples based on real-world data.

Contents

1. Introduction

In a world of increasing complexity the visualization of data becomes more and more important. Especially in fields such as data mining and social network analysis, a graphical representation provides a far more effective way for humans to understand relationships and find patterns than the raw data in text form would do.

An important technique when it comes to analyzing large multivariate datasets – but also graphs – is the concept of clustering. A clustering classifies the data points of the dataset into groups, the clusters, such that each element is contained in exactly one cluster. Figure 1.1 shows two different clusterings on a set.

$$C_1 = \{\{A, B, C\}, \{D, E\}, \{F, G\}\}$$
$$C_2 = \{\{A\}, \{B, D, F\}, \{C\}, \{E, G\}\}$$

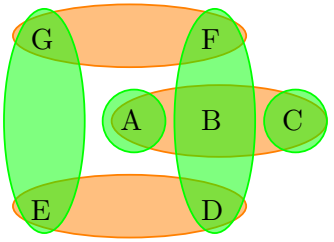


Figure 1.1.: Two clusterings on the set $\{A, B, C, D, E, F, G\}$ with a simultaneous embedding

Usually, clusters are based on the similarity (or connectedness, in the case of a graph) of the elements in the dataset. Therefore, elements in the same cluster are elements that are related to each other.

Clusterings can be either intrinsic to the datasets or result from a clustering method.

In an intrinsic clustering, the clusters are implicitly given by one of the dimensions of the data set. In the example of a data set containing information about a group of people, an intrinsic clustering could be the birthplace of a person.

Clustering methods in the field of data mining usually try to aggregate points to a cluster that have a low distance from each other in the multidimensional space of the attributes. For network analysis, metrics for the connectedness of the nodes can be used to cluster the nodes into highly connected groups.

However, there often exist multiple intrinsic clusterings of a dataset. And also clustering methods can provide different results depending on the algorithm or the parameters used.

This raises the problem of comparing two different clusterings.

Indices such as the Rand index [Ran71] provide an objective numerical value describing the similarity of two clusterings, but they provide no information on how single clusters actually relate to each other.

Contrary to that, textual representations of clusterings as lists or tables provide detailed information on the containing clusters for every single data point. These representations are useful for a detailed analysis but fail to provide a general overview of the situation, even when they are enriched with different colors (see [DCS06]).

In this work we will focus on a visual comparison of clusterings.

We can visualize a single clustering by defining a region in the plane for each cluster and placing the contained data points of the cluster inside its region. We call this an *embedding* of the clustering. When we repeat this for a second clustering, we can compare the two drawings. However, since the data points might have been placed at different locations, finding the same point in both drawings can be difficult.

This leads us to our main question: how can we simultaneously embed two different clusterings in the same drawing?

A visualization of such an embedding would allow an intuitive comparison of the results of clustering methods. For intrinsic clusterings, it could reveal differences and relationships between different dimensions of the data. Furthermore, it can help to visually evaluate the quality of a clustering method compared to a given intrinsic clustering. A simple example how such an embedding might look like is given in Fig. 1.1.

1.1. Related Work

Previous works on embeddings of multiple sets focus either on a more restricted or a more general version of our problem.

In the restricted case, multiple clusterings have to be hierarchical. In the general case, the sets to be embedded are not limited to the structure of a clustering, and an element can be contained in an arbitrary number of sets.

Examples for works on hierarchical clusterings include approaches for graphs (see [EF97]) and general sets (see e.g. [HY00]). Also embeddings of a single clustering have been studied in relation with drawings that include edges of graphs. A core concept in this field is *clustered planarity* (see [FCE95]).

Visualizations of intersections between subsets in general have first been used by Euler [EdCCC42] in the 18th century and later been modified and formalized by Venn [Ven80]. In more recent works, these concepts have been refined and extended (see [Cho07] for an overview). An interesting extension to this topic are spider-diagrams (see e.g. [GHK99]) where a graph is drawn on top of an existing Euler diagram. These diagrams can be used for logical reasoning systems.

Interesting approaches for embedding a collection of subsets can also be found in literature on hypergraphs and their planarity (see [KvKS09] for an overview). More practical approaches such as convex-hull-representations [SAA09], “Bubble Sets” [CPC09], and “Line Sets” [ARRC11] also aim to highlight general subsets and their relationships.

Nevertheless, all these approaches for embeddings focus either on the special case of hierarchical clusterings or on subsets in general. When applying the methods for general representations of subsets induced by clusterings, the structure of the clusterings is not taken into account. This can result in undesirable intersections between the regions of the same clustering. To our knowledge, there exist no publications that study embeddings of subsets for the special case that these are induced by two or more clusterings.

1.2. Our Contribution

In this work we are going to develop the theoretical background for the simultaneous visualization of two (or more) clusterings. We are going to focus on embeddings of the clusterings and therewith provide the foundations for drawings that fulfill certain criteria to increase readability. These criteria mainly focus on avoiding intersections between the regions of two clusters, especially if these clusters do not share a common vertex.

After an overview of basic principles and definitions that will be of importance for this work (Chapter 2), we will introduce concepts of modeling the relationships between clusters in Chapter 3.

In Chapter 4 we focus on different levels of embeddability and how they relate to known combinatorial problems from other fields. We establish a hierarchy that contains eight different classes of embeddability. With one exception, we can provide examples that show that the classes are true subsets of each other and thus prove the strictness of the hierarchy.

We furthermore examine the complexity of the decision problems whether an instance belongs to a class in the hierarchy or not. For most classes we can use the results from related concepts. For a central concept, strong embeddability, this is not possible, and we thus provide our own proof to show the $\mathcal{NP}$-completeness of the corresponding decision problem.

In Chapter 5 we provide practical approaches to generate embeddings of two clusterings. We focus on grid representations of clusterings that are derived from two of the classes in the hierarchy. The decision problem whether an embedding of this type is optimal is also $\mathcal{NP}$-complete. We provide an approach to generate such an optimal solution with an integer linear program. In addition, we develop a fast heuristic that generates an approximation of the optimal solution.

In the last chapter we conduct experiments with the heuristic on randomly generated instances of grid representations. We examine the distribution of crossings between clusters depending on the similarity of the clusterings. We further experimentally evaluate the results of the heuristic against the optimal solution generated by the integer linear program and apply our method to two instances that are based on real-world data.

We conclude the work with an overview and an outlook on open problems and future work.

2. Graphs, Hypergraphs, and More

In this chapter we present some fundamental concepts that are the basis to the ideas we will develop in this work. We begin by introducing graphs and their embeddings and then extend the concept to hypergraphs and some related ideas.

2.1. Graphs, Curves, and Embeddings

Graphs are a fundamental concept used in many areas of mathematics and computer science. They are used to model networks and to describe the relationship between elements of a set. We are particularly interested in planar graphs. The book of Nishizeki and Chiba [NC88] gives a good overview on the theory of planar graphs and related algorithms. In the following we will briefly discuss the graph-theoretical concepts that are relevant to this work.

An (undirected) *graph* $G = (V, E)$ consists of a set of nodes V and a set of edges E, where an edge $e \in E$ is a pair $e = \{u, v\}$ with $u, v \in V$. We say that u and v are *incident* to e and vice versa. Two nodes are *adjacent* if they are connected by an edge, and two edges are *adjacent* if they share a common node. A *path* from v_1 to v_n, $v_1, v_n \in V$ in G is an alternating sequence $v_1, e_1, v_2 \dots v_{n-1}, e_{n-1}, v_n$ of edges and nodes in G, where $e_i = \{v_i, v_{i+1}\}$ and the nodes $v_2 \dots v_n$ are distinct. The *length* of a path is the number of edges in the path. A path from v_1 to v_n is called *circle* if $v_1 = v_n$. A graph is *connected* if there exists a path between any two nodes u and v in V.

A graph is *simple* if it does not contain any parallel edges and loops (circles of length one). Unless explicitly stated, all graphs in this work are assumed to be simple. If a graph is not simple, we call it a *multigraph*.

We are going to introduce embeddings of graphs in the plane. In order to do this, we first need to define curves in the plane:

2.1.1. Curves in the Plane

A *curve* γ in the plane is a continuous map $\gamma : I \to \mathbb{R}^2$ from an interval $I = [a, b] \subset \mathbb{R}$ to the plane $\mathbb{R}^2$. The points $\gamma(a)$ and $\gamma(b)$ are called the *endpoints* of the curve.

A curve is *simple* if it does not have any internal self-intersections:

$$\forall x, y \in I \setminus \{a, b\} : \ \gamma(x) = \gamma(y) \Rightarrow x = y$$

A curve is *closed* if both of its endpoints are mapped to the same point:

$$\gamma(a) = \gamma(b)$$

A curve C_a in the plane is *homeomorphic* to another curve C_b if there exists a continuous bijection from the points in C_a to the points C_b, whose inverse is also continuous.

A *Jordan curve* J is a simple closed curve in the plane, i.e., a non-self-intersecting, continuous loop. It is homeomorphic to a circle in the plane. The Jordan theorem [Jor87], proved in 1887, states that every Jordan curve divides $\mathbb{R}^2$ into a bounded and an unbounded region. We define $int(J)$ and $ext(J)$, respectively, to describe the subsets of $\mathbb{R}^2$ corresponding to these regions. We will also use the intuitive notions of inside and outside of the curve. To remain consistent with some later definitions, we define in this work that the points on the curve J itself belong to the inside $int(J)$, i.e., $int(J) = \mathbb{R}^2 \backslash ext(J)$.

A set of Jordan curves $\mathcal{J} = \{J_1, \ldots, J_n\}$ divides the plane into a set of disjoint *regions* $\mathcal{R}(\mathcal{J})$. Mathematically, we can define the regions as follows:

$$\mathcal{R}(\mathcal{J}) := \{X_1 \cap X_2 \cap \ldots \cap X_n \mid X_i \in \{int(J_i), ext(J_i)\}\}$$

For a region $R \in \mathcal{R}$ in the plane, we denote by $comp(R) := \{R_1, \ldots R_n\}$ the set of the (path-) *connected components* of the region. These are the subsets of the region, for which there exists a curve C between any two points in the subset such that C lies completely within the subset.

We define the *external region* $ext(\mathcal{J}) \subset \mathbb{R}^2$ of the set of Jordan curves as the, not necessarily connected, region that lies on the outside of all Jordan curves:

$$ext(\mathcal{J}) = \bigcap_{J \in \mathcal{J}} ext(J)$$

In later chapters when it comes to embeddings of clusterings we will refer to the *faces* $F(\mathcal{J})$ of a set of Jordan curves $\mathcal{J}$. These are simply the connected components of the external region:

$$F(\mathcal{J}) = comp(ext(\mathcal{J}))$$

2.1.2. Embeddings and Planarity of Graphs

An embedding $\Pi(G) = (P(V), C(E))$ of a graph $G = (V, E)$ in the plane consists of a set of points $P(V)$ with a point $p(v) \in \mathbb{R}^2$ for every node $v \in V$ and a set of simple curves $C(E) \subset \mathbb{R}^2$ with a curve $c(e)$ for every edge $e = \{u, v\}$. The endpoints of $c(e)$ correspond to the points $p(v)$ and $p(u)$ of the incident nodes u and v.

A graph G is *planar* if it has a *planar embedding* where all edges are disjoint except for their endpoints. It is possible to determine in linear time if a graph is planar: Hopcroft and Tarjan [HT74] proposed the first efficient algorithm in 1974. The curves in planar embeddings divide the plane into regions that are called the *faces* of the embedding. A node or an edge is *incident* to a face if it lies on its boundary, and two faces are *adjacent* if they are incident to a common edge.

Let $\Pi(G)$ be a planar embedding of G and F the set of faces. The dual graph $G^* = (V^*, E^*)$ that is induced by $\Pi(G)$ has a node v_f in V^* for each face f in F and an edge between two nodes v_{f_1}, v_{f_2} if the corresponding faces $f_1, f_2 \in F$ of $\Pi(G)$ are adjacent.

A combinatorial planar embedding $\Pi^c(G)$ of a planar graph $G = (V, E)$ is a *rotation system*, i.e., a cyclic order $\pi_v = [e_1, \ldots e_n]$ of the incident edges of every node $v \in V$ that

induces a planar embedding. Two planar embeddings of G are *equivalent* if they induce the same combinatorial planar embedding. A graph G has a *unique planar embedding* if all embeddings of G are equivalent, i.e., if there exists only one rotation system that induces a planar drawing.

A *subdivision* G' of a graph G is a graph, where one or more edges of G have been replaced by a path via newly added nodes.

A graph is *3-vertex-connected* (or just *3-connected*) if the graph is still connected after any two nodes are removed.

A graph has a unique planar embedding if it is a subdivision of a 3-connected planar graph (See [NC88], Theorem 1.1).

2.1.3. Classes of Graphs and Substructures

A graph $T = (V, E)$ is a *tree* if it is connected and has no circles. All trees are planar. A tree has exactly $|V| - 1$ edges and any planar embedding of a tree has only one face. A graph consisting of multiple trees is a *forest*.

The *induced subgraph* $G[W]$ of a subset $W \subset V$ of the nodes of a graph $G = (V, E)$ is the graph $G[W] = (W, E')$ where $E' = \{\{u, v\} \in E \mid u \in W \wedge v \in W\}$ is the subset of the edges with both endpoints in W.

A graph is called *k-partite* if there exists a partition of its nodes into k *disjoint sets*, such that no two adjacent nodes are in the same set. For $k = 2$ we also use the term *bipartite*. If $V = V_a \cup V_b$ and $V_a \cap V_b = \emptyset$, we use the notation $G = (V_a, V_b, E)$ for a bipartite graph on the partitions V_a and V_b.

A *k-clique* in a graph is a set of k nodes that induce a subgraph in which all nodes are pairwise connected. We call a graph with k nodes *complete* if it is a k-clique. We write K_k for the complete graph with k nodes. The complete graph with five nodes, K_5 is one of the two fundamental non-planar graphs. The other one is $K_{3,3}$, the *complete bipartite graph* on two sets of three nodes each. Complete bipartite means that each node is connected to all nodes in the other partition.

A node is in the *k-core* of G if it is connected to at least k nodes that are also in the k-core. Thus the k-core is the subgraph of G that remains after successive removal of nodes of degree less than k.

2.2. Hypergraphs and Hypergraph-Planarity

A *hypergraph* is a generalization of the concept of graphs, where edges are not limited to connect exactly two nodes, but are allowed to link an arbitrary set of vertices. Referring to the book from Berge [Ber89], we give the following formal definition:

Let 2^V be the power set of V. A hypergraph $H = (V, \mathcal{S})$ consists of a set of vertices V and a set $\mathcal{S}$ of *hyperedges*, which are non-empty subsets of V that cover all vertices in V:

$$\mathcal{S} \subseteq 2^V, \bigcup_{S \in \mathcal{S}} S = V$$

We call $v \in V$ *incident* to $S \in \mathcal{S}$ (and vice versa) if $V \in S$. The *degree* $d_H(v)$ of a vertex $v \in V$ is defined as the number of edges incident to v: $d_H(v) = |\{S \in \mathcal{S} \mid v \in S\}|$

If all vertices in the hypergraph H have the same degree k, we call H *k-regular*. The hypergraph H is *complete* if the set of hyperedges corresponds to the power set of the vertices ($\mathcal{S} = 2^V$), and *linear* if every pair of sets shares at most one vertex.

For the visualization of hypergraphs, it is often useful to aggregate vertices that share the same hyperedges to a single vertex. Mäkinen [Mäk90] has defined a binary equivalence relation: two vertices $u, v \in V$ are *equivalent* if they are incident to the same set of hyperedges. The *condensation* H' of a hypergraph H is the hypergraph that results if we identify equivalent vertices with one *representing vertex* from each equivalence class. We will call a hypergraph *condensed* if it is equal to its condensation.

In the literature on graph-theory the terms vertex and node are usually used interchangeably. In this work however, we will try to maintain a nomenclature, in which we use the term *node* for an element that is part of an ordinary graph. The term *vertex* is reserved for elements in sets, such as the elements of a hypergraph. This will hopefully make it easier to distinguish between the two concepts in some of the later proofs.

2.2.1. Visualization of Hypergraphs

There are multiple ways to visualize a hypergraph. The most common methods are the *subset standard* and the *edge standard* [Mäk90].

In the edge standard, hyperedges are drawn as a set of curves, called *bunches*, such that two curves in the same bunch are overlapping in the area close to their common vertex (Fig. 2.1(a)). The subset standard encloses vertices within a hyperedge by a Jordan curve (Fig. 2.1(b)). The latter method of visualizing the relationships between sets is related to the concept of *Venn-* and *Euler-diagrams* (See Section 2.3.1 and 2.3.2). In this work we will refer to drawings in the subset standard as embeddings.

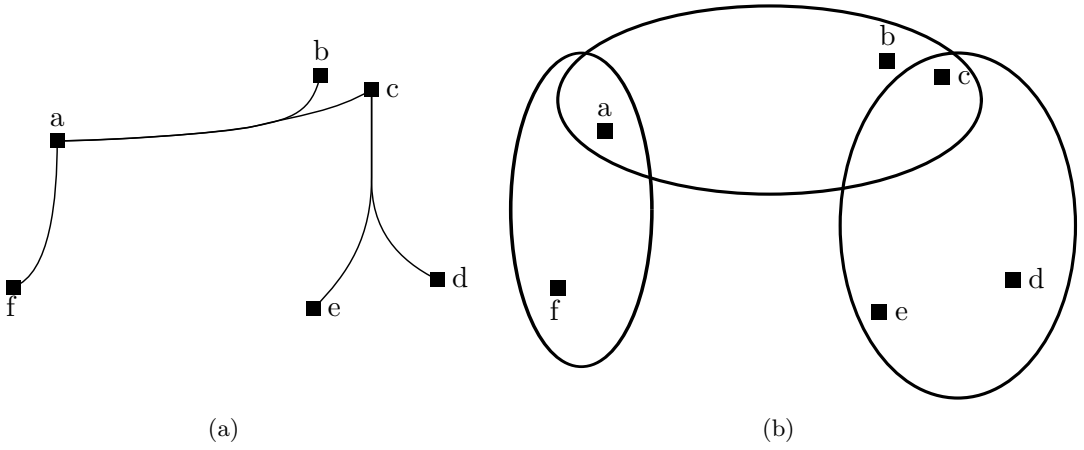


Figure 2.1.: The visualization of a hypergraph in the edge- and in the subset-standard. The example was taken from [Mäk90]

The known concept of planarity for graphs does not transfer to hypergraphs in a straight forward way and scientific literature provides multiple definitions of planarity for hypergraphs. We will now introduce the two concepts that are relevant for this work.

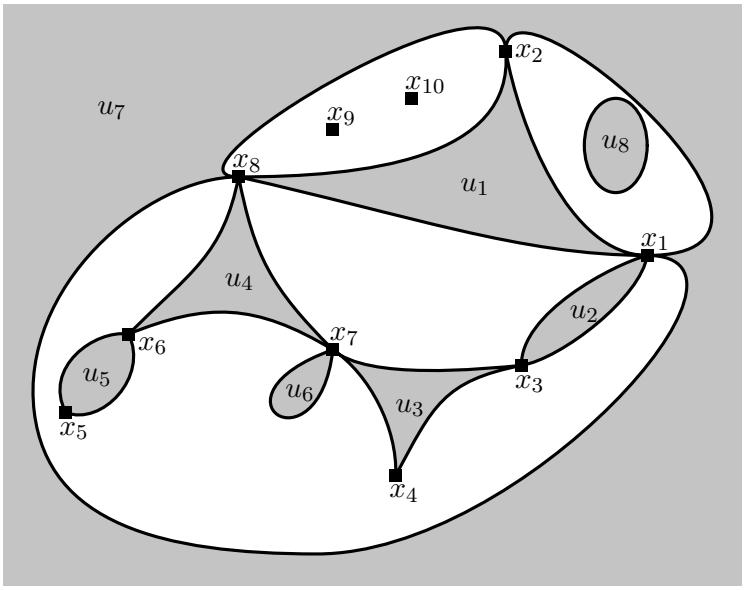
2.2.2. Zykov-Planarity

In Zykov's concept of planarity [Zyk74], hyperedges are represented as faces of a subdivision of the plane with the vertices placed on their boundaries. A graph is Zykov planar if there exists such a subdivision, in which every vertex lies on the boundary of all the faces of its incident hyperedges.

Formally, a hypergraph $H = (V, \mathcal{S})$ is Zykov-planar if there exists a planar embedding $\Pi(G_M)$ of a multigraph $G_M = (V, E_M)$ with the set of vertices of H as nodes and a set of faces F_M, such that:

- every hyperedge $S_i \in \mathcal{S}$ can be mapped to a face $f_i \in F_M$
- every $v \in S_i$ is incident to f_i

Figure 2.2 shows an example of such an embedding for an example-hypergraph[1].



$$
\begin{aligned}
V &= \{x_1, \dots x_{10}\} \\
\mathcal{S} &= \{ \\
&\quad u_1 = \{x_1, x_2, x_8\} \\
&\quad u_2 = \{x_1, x_3\} \\
&\quad u_3 = \{x_3, x_4, x_7\} \\
&\quad u_4 = \{x_6, x_7, x_8\} \\
&\quad u_5 = \{x_5, x_6\} \\
&\quad u_6 = \{x_7\} \\
&\quad u_7 = \{x_1, x_2, x_8\} \\
&\quad u_8 = \{\} \\
&\}
\end{aligned}
$$

Figure 2.2.: An embedding of G_M that shows the Zykov-planarity of the example-hypergraph H. This example is taken from Zykov's original paper [Zyk74]

Zykov planarity is a true generalization of planarity for ordinary graphs: an ordinary graph viewed as a hypergraph is Zykov planar if and only if it is planar in the ordinary sense.

Walsh [Wal75] showed the following relationship between Zykov-planarity and the planarity of bipartite graphs:

A hypergraph $H = (V, \mathcal{S})$ is Zykov planar if and only if there exists a planar bipartite graph $G_b(H) = (V, V(\mathcal{S}), E_b)$, such that:

- $V(\mathcal{S})$ contains a node $v(S)$ for every hyperedge in $\mathcal{S}$
- E_b contains an edge between a node $v \in V$ and $v(S) \in V(\mathcal{S})$ if v is incident to S:

$$E_p = \Big\{ e = \{v, v(S)\} \mid v \in S \Big\}$$

We call $G_b(H)$ the *bipartite map* of H.

It is thus possible to test in linear time (regarding the number of vertices) if a hypergraph H is Zykov planar: we just have to test if its bipartite map G_b is planar.

2.2.3. Vertex Planarity

Johnson and Pollak [JP87] described two other concepts for planarity of hypergraphs: hyperedge planarity and vertex planarity. We will limit this overview to vertex planarity, which turned out to be the more intuitive and – for our purposes – more useful approach.

A hypergraph $H = (V, \mathcal{S})$ is representable by a *vertex-based Venn diagram*[2] if there exists a planar embedding $\Pi(G)$ of a graph $G = (W, E)$ with faces F and a one-to-one mapping

[1] The example was taken from Zykov's original paper [Zyk74]. Note that, contrary to our definition, Zykov allows empty hyperedges (u_8) and *singletons* (x_9 and x_{10} are not incident to any hyperedge)

[2] Note that this concept is based on a definition of *Venn diagrams* that allows empty regions. They correspond rather to Euler diagrams and are not to be confounded with the concept of *Venn diagrams* we are going to define later in this work (Section 2.3.1).

$\Phi : V \to F$ from V to F such that for every hyperedge $S \in \mathcal{S}$ the union of the faces $\Phi(S)$ corresponding to the vertices in S is an interior-connected region of the plane.

A hypergraph is *vertex planar* if it can be represented by a vertex-based Venn diagram.

Vertex planarity also is a true generalization of the known planarity concept: a graph is planar in the ordinary sense if and only if it is vertex planar when viewed as a hypergraph.

However, vertex planarity is less strict than Zykov planarity: if a hypergraph is Zykov-planar, then it is vertex planar. Yet, not every vertex planar hypergraph is Zykov planar.

Unfortunately, the test for vertex planarity is $\mathcal{NP}$-complete, as shown by Johnson and Pollack [JP87].

The concept of vertex planarity naturally leads to the definition of the support of a hypergraph:

2.2.4. The Support of a Hypergraph

To make the definition of vertex planarity easier to handle, Kaufmann et al. [KvKS09] propose the following concept:

A graph $G_p = (V, E_p)$ is a *support* of a hypergraph $H = (V, \mathcal{S})$ if for every $S \in \mathcal{S}$ the subgraph $G[S] = (S, E_S)$ of G that is induced by the vertices in S is connected.

A hypergraph has a planar support G_p if and only if it is vertex planar: G_p is a subgraph of the dual graph of the vertex-based Venn diagram $G = (W, E)$ we defined earlier.

This means that testing if a hypergraph has a planar support is also $\mathcal{NP}$-complete. Buchin et al. [BvKM$^+$10] showed that the problem remains $\mathcal{NP}$-complete even when it is limited to the decision whether a hypergraph has a 2-outerplanar support.

However, there exist polynomial time algorithms to test if a hypergraph has a planar support that is a path, a tree or a cycle. (See [BvKM$^+$10] for an overview.) Verroust-Blondet and Viaud [VBV04] show that every hypergraph with less than nine hyperedges is vertex planar and thus has a planar support. Brandes et al. [BCPS11a] found a polynomial-time method to decide whether a hypergraph has a support that is a cactus (see the paper for a definition of cactus-graphs).

2.3. Related Concepts

Stirling C. Chow gives an overview of related concepts to hypergraphs in Chapter 2 of his dissertation [Cho07]. We will now briefly discuss the most relevant concepts for this work.

2.3.1. Venn Diagrams

A Venn diagram is a method to visualize the relationships between sets and was first introduced by John Venn in 1880 [Ven80]. It represents a set through a Jordan curve in the plane and intersections between sets as intersections between the insides of the curves. The definition requires that all possible intersections between the sets have a non-empty region in the plane. Thus, a Venn diagram of n sets divides the plane into 2^n different regions. Regions that represent empty sets can be shaded in gray.

Venn diagrams are well studied, and Venn himself proved in his original article that there exists a Venn diagram for any number of sets. Unfortunately, any visualization of a Venn diagram of more than 4 sets becomes very complex. Also, for the types of hypergraphs we are going to study, only a small fraction of the intersections is going to be non-empty. This leads to the related concept of Euler diagrams.

2.3.2. Euler Diagrams

Leonard Euler first used the diagrams that are named after him in his "Lettres a une princesse d'Allemagne" [EdCCC42], first published in 1768. Euler diagrams are a generalization of Venn diagrams, allowing empty regions. Since Euler did not give a formal definition, there exist different formal definitions in scientific literature. We will use the definition from the article by Chow and Ruskey [CR04]:

A set of Jordan curves $\mathcal{J}$ forms an Euler diagram if every non-empty region $R \in \mathcal{R}(\mathcal{J})$ is connected.

An Euler diagram (and by extension a Venn diagram) is called *simple* if a maximum of two Jordan curves intersect at any given point.

2.3.3. String Graphs

String graphs were first used by Benzer [Ben59] and Sinden [Sin66] in works on genetic structures and layout of thin film transistors, respectively. Ehrlich et al. [EET76] later formalized the concept:

A graph $G = (V, E)$ is a *string graph* if there exists a curve in the plane for every node, such that two curves intersect if and only if their corresponding nodes are connected.

A *representation* $R = \{R(v), v \in V\}$ of G is a collection of curves in the plane with $R(v) \cap R(u) \neq \emptyset$ if and only if $\{u, v\} \in E$.

Every planar graph is a string graph. If a graph $G' = (V', E')$ is constructed from a graph $G = (V, E)$ by subdividing every edge, G' is a string graph if and only if G is planar.

Until rather recently it was not even clear if the problem whether an arbitrary graph is a string graph is decidable. Finally, Pach and Tóth [PT02] and Schaefer and Štefankovič [Su04] independently proved an exponential upper bound for the number of crossings, and shortly afterwards, Schaefer et al. showed the $\mathcal{NP}$-completeness [SSu03].

In 2009, Chalopin and Gonçalves [CG09] proved Scheinerman's conjecture [Sch84] that every planar graph has a representation as a string graph with straight line segments as curves.

A graph has a *1-string representation* if it is a string graph, where each pair of strings is allowed to intersect at most once [CGO10].

For our work, we are particularly interested in a special kind of string graphs:

A bipartite graph $G = (V_a, V_b, E)$ has a *grid representation* if the nodes in V_a can be represented as horizontal and the nodes in V_b as vertical disjoint line segments in the plane, such that two line segments intersect if and only if their corresponding nodes are connected. We call such graphs *grid intersection graphs*.

The concept of grid intersection graphs has first been introduced by Hartman et al. [HNZ91]. They also showed that every planar bipartite graph is a grid intersection graph. Kratochvíl showed that the problem to decide if an arbitrary bipartite graph is a grid intersection graph is $\mathcal{NP}$-complete [Kra94].

In a subsequent paper in cooperation with Przytycka [KP96], Kratochvíl examines grid intersection graphs on the annulus and torus. In both cases the problem of deciding whether a graph is a grid intersection graph remains $\mathcal{NP}$-complete. In the following we will refer to the grid intersection graph on the annulus as *cylindrical grid intersection graph*.

3. Families of Clusterings

We will now define concepts for describing the relationships between clusters from different clusterings on the same set. Before we do that, we provide a formal definition of clusterings:

A *clustering* $\mathcal{C} = \{C_1 \dots C_n\}$ on a set of *vertices* V is a collection of n sets $C_j \subseteq V,\ j = 1, \dots, n$. We call $\mathcal{C}$ *strict partitioning clustering* if each vertex belongs to exactly one cluster:

$$\bigcup_{i=1}^{n} C_i = V$$
$$C_i \cap C_j = \emptyset,\ 1 \leq i < j \leq n$$

In the following chapters all clusterings are assumed to be strict partitioning clusterings.

From now on let V be a set of vertices and $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ a family of k mutually different clusterings on V. To simplify future definitions, we will often just talk about a family of clusterings $\mathcal{F}$ without explicitly mentioning the set V. The clusterings implicitly describe V as the union of all clusters.

3.1. Representation of Multiple Clusterings

We begin by defining the cluster-graph that helps us to model relationships between multiple clusterings. In Section 3.2 we will provide a complete characterization of the properties of cluster-graphs.

Definition 1 (Cluster-Graph) *The* cluster-graph $G_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ *of* $\mathcal{F}$ *has a* corresponding node $v(C) \in V_{\mathcal{F}}$ *for each cluster* $C \in \bigcup_{\mathcal{C} \in \mathcal{F}} \mathcal{C}$.

Two nodes $v(C_a), v(C_b) \in V_{\mathcal{F}}$ *are connected by an edge in* $E_{\mathcal{F}}$ *if the two corresponding clusters* C_a *and* C_b *have at least one vertex* $v \in V$ *in common:*

$$E_{\mathcal{F}} = \Big\{ e = \{v(C_a), v(C_b)\} \mid C_a \cap C_b \neq \emptyset \Big\}$$

Figure 3.1 shows an example of a set with two clusterings and the corresponding cluster-graph.

An alternative way to model the structure of multiple clusterings on a set is by representing it as a hypergraph:

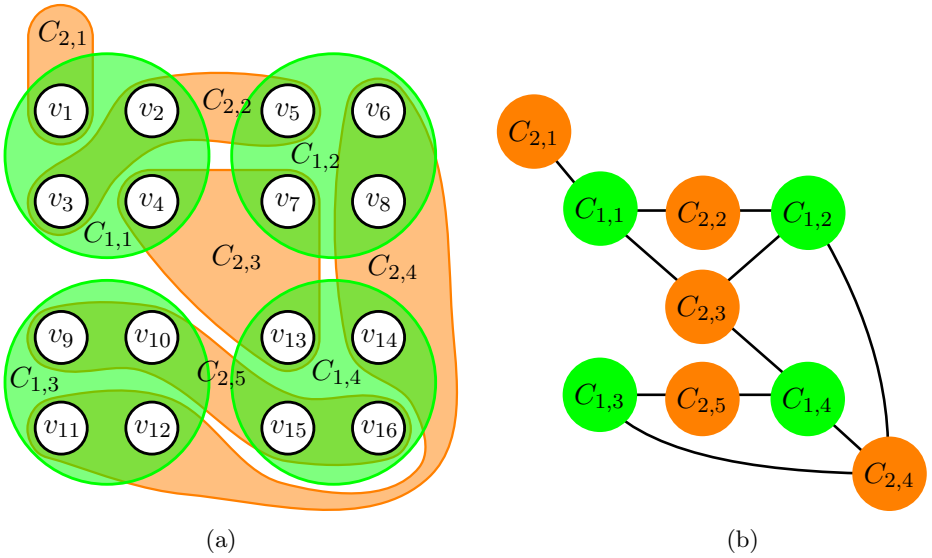


Figure 3.1.: An embedding of two different clusterings on a set and the resulting cluster-graph

Definition 2 (Corresponding Hypergraph) *The* corresponding hypergraph $H_{\mathcal{F}} = (V, \mathcal{S}_{\mathcal{F}})$ *of* $\mathcal{F}$ *is the hypergraph on the set of vertices* V *that uses the union of the clusters from all clusterings in* $\mathcal{F}$ *as hyperedges:*

$$\mathcal{S}_{\mathcal{F}} = \bigcup_{\mathcal{C} \in \mathcal{F}} \mathcal{C}$$

Every corresponding hypergraph $H_{\mathcal{F}}$ is k-regular, since every vertex is connected to exactly k hyperedges. For two clusterings the condensed hypergraph $H'_{\mathcal{F}}$ is also linear. In the following chapters it will be useful to adapt the concept of a support to families of clusterings:

Definition 3 (Support of a Family of Clusterings) *A graph* $G_p = (V, E_p)$ *is a* support *of* $\mathcal{F}$ *if it is a support for the corresponding hypergraph* $H_{\mathcal{F}}$.

In order to be able to make statements about such supports, we will often refer to supports that are minimal:

Definition 4 (Minimal Support) *A support* $G_p = (V, E_p)$ *of* $\mathcal{F}$ *is* minimal *if no edge in* E_p *can be removed without destroying the support-property. We call a support* $G'_p = (V, E'_p)$ *of* $\mathcal{F}$ *a* minimalization *of a non-minimal support* $G_p = (V, E_p)$ *of* $\mathcal{F}$ *if* G'_p *is minimal and* $E'_p \subset E_p$.

Minimal supports have an interesting property in the case of two clusterings that we will be using later to generate embeddings from supports:

Proposition 1 *If* $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ *consists of two clusterings, the induced subgraph* $G_p[C]$ *of every cluster* C *in a minimal support* G_p *of* $\mathcal{F}$ *is a tree.*

Proof

Suppose there exists a cluster C_a in a minimal support G_p, such that $G_p[C_a]$ is not a tree. Without loss of generality, let C_a be a cluster in the clustering $\mathcal{C}_1$. There exists thus a circle in C_a. Let $C \subseteq C_a$ be the vertices on the circle. Let now $C_b \in \mathcal{C}_2$ be an arbitrary cluster in the other clustering that shares a vertex with C. There are two possibilities: either C is completely contained in C_b or C_b contains only a subset of C. In the first case we can remove an arbitrary edge from the circle without loosing the connectedness-property of $G_p[C_a]$ and $G_p[C_b]$. Thus, G_p is not minimal. In the second case there exists a vertex u with $u \in C$ and $u \in C_b$ that has an edge $e = \{u, v\}$ to a vertex $v \in C$ with $v \notin C_b$. We can remove e without destroying the connectedness of the support in any of the three involved clusters. Thus, G_p can not be a minimal support either. This is a contradiction to the initial assumption. □

The intersection graph between the sets of a hypergraph, which we refer to as cluster-graph, is also called line-graph or representive graph in literature [Ber89]. See [MM99] for an extensive overview on the theory of intersection graphs.

3.2. Properties of the Cluster-Graph

In this section we will focus on the properties of cluster-graphs. Before we examine their structure, we define the following concept:

Definition 5 (k-Clique-Graph) *A graph $G = (V, E)$ is a* k-clique-graph[1] *if it consists only of k-cliques. That is, for every edge $\{u, v\} \in E$ there exists a k-clique $K \subset V$ with $u, v \in K$.*

Now we can completely characterize cluster-graphs with the following two theorems:

Theorem 1 *The cluster-graph $G_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ of a family of k clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ on a set V is k-partite and a k-clique-graph.*

Proof

Let us assume that the cluster-graph $G_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ is not k-partite. Then there are at most $k - 1$ disjoint sets in $G_{\mathcal{F}}$. This means that there must be at least one clustering $\mathcal{C} \in \mathcal{F}$, whose set of corresponding nodes $V(\mathcal{C}) \subset V_{\mathcal{F}}$ in the cluster-graph is not disjoint. Thus, there exist two nodes $v(C_a), v(C_b) \in V(\mathcal{C})$ that are connected by an edge $e = \{v(C_a), v(C_b)\}$. By the definition of the cluster-graph the edge e implies that C_a and C_b share a common vertex: $C_a \cap C_b \neq \emptyset$. Since C_a and C_b belong to the same clustering, $\mathcal{C}$ can not be a strict partitioning clustering. This is a contradiction to our definition of clusterings.

Let $e = \{v(C_a), v(C_b)\} \in E_{\mathcal{F}}$ be an edge in the cluster-graph. Thus, there exists a vertex $v \in V$ with $v \in C_a$ and $v \in C_b$. Since all k clusterings in $\mathcal{F}$ are strict partitioning clusterings, v is in exactly one cluster of every clustering: $\forall \mathcal{C} \in \mathcal{F} : \exists! C \in \mathcal{C} : v \in C$. Thus, there are k clusters sharing v as a common vertex, requiring that their corresponding nodes in the cluster-graph are pairwise connected: they form a clique in $G_{\mathcal{F}}$. Thus, every edge is in a k-clique and $G_{\mathcal{F}}$ a k-clique-graph. □

Theorem 2 *For every k-partite k-clique-graph $G = (W, E)$ there exists a family of k clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ on a set V, such that G is the cluster-graph of the clusterings in $\mathcal{F}$.*

[1] Note that this definition is not coherent with the definition in [RS69] where a clique-graph is an intersection-graph between cliques of another graph

Proof (by construction)

We construct V and the corresponding clusterings $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ as follows:

Since G is k-partite, we can color the nodes in W in k different colors, one for each partition. We construct a clustering $\mathcal{C}_i$ for every color i, containing a corresponding cluster $C_w \in \mathcal{C}_i$ for every node $w \in W$ of color i. We now create a vertex $v_K \in V$ for every k-clique $K \subseteq W$ in G. For every node $w \in K$ in the clique we add the vertex v_K to the corresponding cluster C_w.

Since every k-clique K has exactly one node of every color (due to the k-partiteness), every vertex $v_K \in V$ is placed in exactly one cluster in every clustering. We have thus constructed a set of strict partitioning clusterings. Since every edge in G is part of a k-clique, we have covered all edges in our construction and thus all intersections between clusters. When reversing the construction and generating the cluster-graph of $\mathcal{F}$, we get exactly G. □

Due to its construction, every family of clusterings $\mathcal{F}$ has exactly one cluster-graph $G_{\mathcal{F}}$. However, a cluster-graph $G_{\mathcal{F}}$ can correspond to several different families of clusterings. Figure 3.2 gives an example: The 3-clique v_5 in the middle of the cluster-graph (Fig. 3.2(a)) does not necessarily imply a vertex in the underlying set since every edge of the clique is already covered by another 3-clique. This allows the two different underlying sets shown in Fig. 3.2(b) and Fig. 3.2(c).

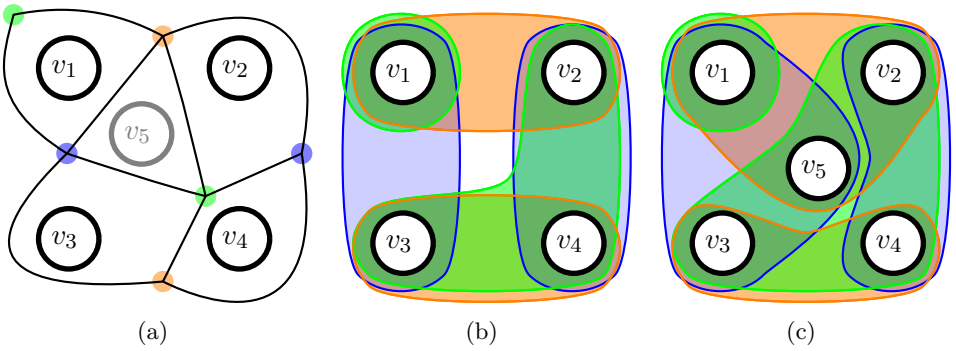


Figure 3.2.: A 3-partite cluster-graph and two different possible underlying sets. Vertices of the underlying set are drawn in the center of their corresponding 3-clique of the cluster-graph

Since every edge in a graph induces a 2-clique, every bipartite graph is a 2-partite 2-clique-graph. Thus, Theorem 2 implies the following corollary:

Corollary 1 *Every bipartite graph $G = (V_a, V_b, E)$ is a cluster-graph of a family of two clusterings.*

4. Simultaneous Embeddability of Clusterings

In this chapter we define different classes of embeddability for families of clusterings. We show how these classes relate to planarity of hypergraphs and the other concepts presented in Chapter 2 and examine their complexity.

We start by formally defining the general concept of an embedding for multiple clusterings. Once again, let $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ be a family of k clusterings on a set V.

Definition 6 (Embeddings) *Given $\mathcal{F}$ and V, we define a set of Jordan curves $\mathcal{J}(\mathcal{F})$ in the plane that contains a curve $J(C)$ for every cluster $C \in \bigcup_{\mathcal{C} \in \mathcal{F}} \mathcal{C}$. We further define a set of points $P(V) \subset \mathbb{R}^2$ that contains a point $p(v)$ in the plane for every vertex $v \in V$. We call $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ an* embedding *of $\mathcal{F}$ if the curve of every cluster encloses exactly the points of its contained vertices:*

$$v \in C \Leftrightarrow p(v) \in int(J(C))$$

An embedding of $\mathcal{F}$ corresponds to a drawing of the corresponding hypergraph $H_{\mathcal{F}}$ in the subset standard. We will now extend the notion of faces of a set of Jordan curves to embeddings. This definition will be useful later when it comes to introducing a combinatorial description of embeddings.

Definition 7 (Faces of Embeddings) *The* faces $F(\Gamma)$ *of an embedding $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ of a family of clusterings $\mathcal{F}$ are the faces of the Jordan curves of the embedding:*

$$F(\Gamma) := F(\mathcal{J}(\mathcal{F})) = comp(ext(\mathcal{J}(\mathcal{F})))$$

We say that a cluster $C \in \bigcup_{\mathcal{C} \in \mathcal{F}} \mathcal{C}$ is incident *to a face $f \in F(\Gamma)$ if the face is bounded by the Jordan curve $J(C)$ of the cluster.*

In the following sections we will introduce various restrictions for the curves of embeddings. This eventually leads us to a hierarchy of embeddability (Section 4.4).

4.1. Weak Embeddability

We start with the weakest concept of embeddability. Let $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ be an embedding of $\mathcal{F}$.

Definition 8 (Weak Embeddings) *The embedding $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ is a* weak embedding *(WE) if the insides of all curves corresponding to clusters of the same clustering are pairwise disjoint:*

$$C_i, C_j \in \mathcal{C} \Rightarrow int(J(C_i)) \cap int(J(C_j)) = \emptyset$$

We call $\mathcal{F}$ weakly embeddable *if there exists a weak embedding for $\mathcal{F}$.*

The disjointness-property of weak embeddings is a step towards improving the readability. It also corresponds to the nature of a strict partitioning clustering. We will now show that this restriction is no real constraint since every family of clusterings $\mathcal{F}$ has a weak embedding.

Theorem 3 *Every family of clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ is weakly embeddable.*

Before we prove the theorem, we will define a useful concept that we will also apply in later proofs:

Definition 9 (Enclosing Curves) *Let $\Pi(T) = (P(V), C(E))$ be a planar embedding of a tree $T = (V, E)$ in a plane. We call a Jordan curve J_T an* enclosing curve with distance d to $\Pi(T)$ *if the curves $C(E)$ lie in the inside of J_T and the distance between J_T and the curves is d:*

$$\begin{aligned} C(E) &\subset int(J_T) \\ \forall p \in J_T : \exists q \in C(E) &: \|p - q\| = d \\ \forall p \in J_T, q \in C(E) &: \|p - q\| \geq d \end{aligned}$$

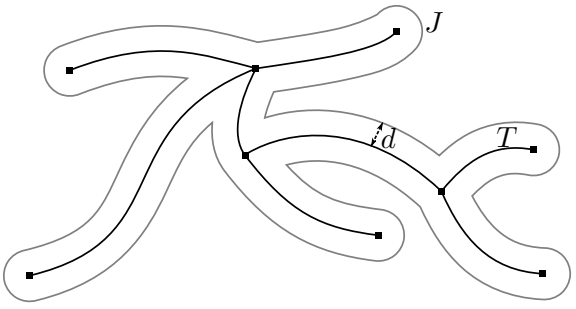


Figure 4.1.: An enclosing curve J_T with distance d to an embedding of a tree T

Figure 4.1 gives an example of an enclosing curve. Since any planar embedding of a tree only has one face, we can always find an enclosing curve if we choose d small enough. This gives us a tool to prove Theorem 3 and several other statements in the upcoming sections of this work.

Proof of Theorem 3

We place all vertices in V at arbitrary – but distinct – locations in the plane. For every cluster C_i of a clustering $\mathcal{C}$ we create an arbitrary spanning tree $T_i = (C_i, E_i)$ with the vertices in C_i as nodes. The union of these trees forms a forest $F = (V, E)$ with the vertices of V as nodes. The forest F has $|V| - |\mathcal{C}| - 1$ edges, is planar and has no circles. Thus, there is only one face in any planar embedding. Therefore, F can be embedded by keeping the locations of the vertices fixed and by drawing the edges of every T_i as disjoint simple curves in the plane. Let d be the minimal distance between the curves from two different trees $T_i, T_j \in F$ and $\epsilon < \frac{1}{2}d$. We can now generate an enclosing Jordan curve $J(C_i)$ for every $C_i \in \mathcal{C}$ with a distance of $\frac{1}{2}d - \epsilon$ to the plane curves of T_i. Thus, all clusters in $\mathcal{C}$ are represented by Jordan curves that have a minimum distance of 2ϵ from each other, guaranteeing their disjointness. The same procedure can be repeated independently for every clustering $\mathcal{F}$ by keeping the vertices fixed. We end up with a weak embedding. □

Example

Let $V = \{v_1, \ldots, v_{10}\}$ be a set and $\mathcal{C} = \{C_1, C_2, C_3\}$ a clustering with $C_1 = \{v_1, \ldots v_6\}$, $C_2 = \{v_7, v_8, v_9\}$ and $C_3 = \{v_{10}\}$.

Figure 4.2 shows a spanning tree for each cluster and a planar embedding of the resulting forest in the plane. The minimal distance d between the curves dictates the maximal thickness of the surrounding Jordan curves. The same construction can be performed independently for each clustering of V and results in a weak embedding of $\mathcal{F}$. Note that the spanning trees correspond to a minimal planar support of $\mathcal{F}$. We will be reusing this technique to generate embeddings from supports in the next sections.

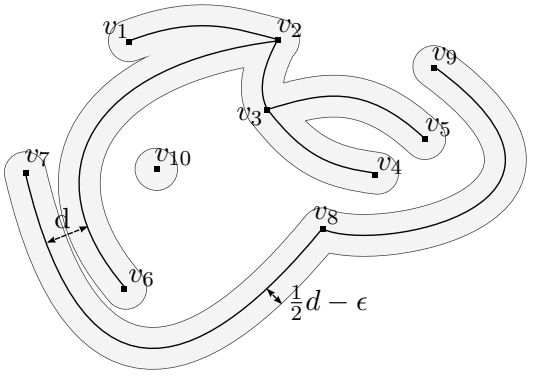
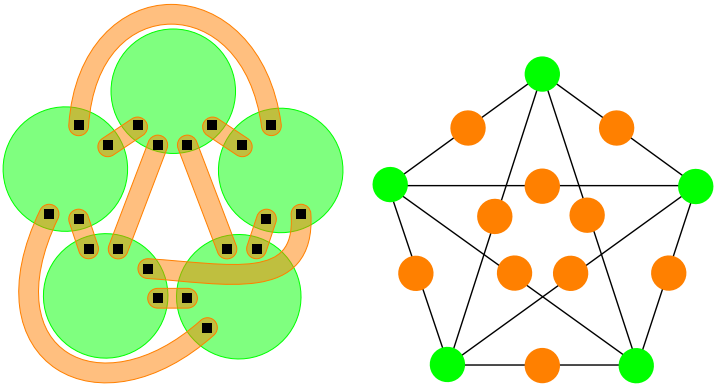


Figure 4.2.: An example how to construct the Jordan curves of clusters from a support that consists of spanning trees

In a weak embedding, the clusters of every clustering are embedded independently. This can result in a curve of one clustering intersecting a curve of another clustering, even if their corresponding clusters do not share a common vertex. Figure 4.3 shows such an intersection: the green cluster on the lower right is cut by an orange cluster, even though they do not share a common vertex. In order to avoid such cuts, we are going to define a stronger form of embeddability in the next section. However, as we will show later, it is impossible for the two clusterings in Fig. 4.3 to be drawn without such an intersection.

For the case of weak embeddings of two clusterings, we can also make the following observation about the regions of the Jordan curves of the embedding: Every region $R \in \mathcal{R}(\mathcal{J}(\mathcal{F}))$ is of one of the following three types. It is in the inside of either zero, one or two curves.

In the first case it is a subset of the external region of the curves and therefore does not contain a vertex. In the second case it is also empty since no vertex is contained only in one



(a) (b)

Figure 4.3.: A weak embedding of a family of two clusterings whose cluster-graph corresponds to a K_5 with subdivided edges

cluster. Only in the third case it can contain a vertex. In the class of embeddability that we are now going to introduce, such a region has to contain a vertex in every connected component.

4.2. Strong Embeddability

We will now introduce a stricter concept of embeddability called strong embeddability. In this concept clusters from different clusterings are not allowed to intersect in an arbitrary way but have to share a common vertex in the areas of intersection:

Definition 10 (Strong Embeddings) *The embedding* $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ *is a* strong embedding *(SE) if it is a weak embedding and every connected component of an intersection of the insides of any two curves* $J(C_a), J(C_b) \in \mathcal{J}(\mathcal{F})$ *contains a vertex:*

$$\forall C_a, C_b \in \bigcup_{\mathcal{C} \in \mathcal{F}} \mathcal{C} : \forall R_i \in comp(int(J(C_a)) \cap int(J(C_b))) : \ \exists v \in V : \ p(v) \in R_i$$

We call $\mathcal{F}$ strongly embeddable *if there exists a strong embedding for* $\mathcal{F}$*.*

In the first section of this chapter, Section 4.2.1, we will look at the relationship of the cluster-graph to string graphs in the case of a strongly embeddable family of clusterings. We will then show how the concept of the 2-core of the cluster-graph allows us to simplify strong embeddings of two clusterings. In the next section, Section 4.2.2, we will show that for the case of two clusterings, strong embeddability is equivalent to vertex planarity. In Section 4.2.3 we will introduce a combinatorial description for strong embeddings and show that some families of two clusterings have unique strong embeddings.

These definitions are a prerequisite for the proof in Section 4.2.4 where we show the $\mathcal{NP}$-completeness of the test for strong embeddability of two clusterings. Finally, in Sections 4.2.5 and 4.2.6, we will introduce some subclasses of strong embeddability.

4.2.1. Strong Embeddability and the Cluster-Graph

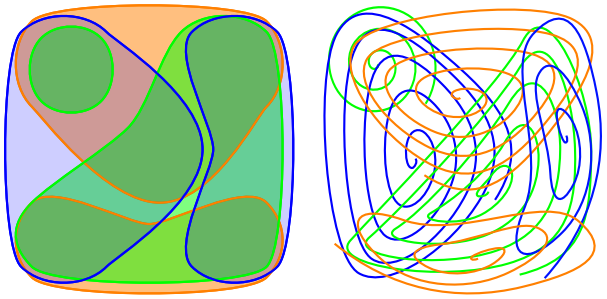
We will begin this section by showing how strong embeddings relate to string graphs:

Lemma 1 *If* $\mathcal{F}$ *is strongly embeddable, then the cluster-graph* $G_{\mathcal{F}}$ *is a string graph.*

Proof

Let $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ be a strong embedding of $\mathcal{F}$. Two clusters C_i, C_j share a common vertex if and only if $int(J(C_i)) \cap int(J(C_j)) \neq \emptyset$. We will have to construct a string (curve) S_i for every C_i such that $S_i \cap S_j \neq \emptyset$ if and only if the corresponding clusters C_i and C_j share a common vertex. In order to do this, we fill the inside of every Jordan curve J_i with a spiral S_i. Such a spiral always exists since every Jordan curve is homeomorphic to a circle, and we can use the same transformation for the circle to transform the spiral (see Fig. 4.4 for an illustration). If the spiral S_i is dense enough, every object in the interior of J_i is intersected by S_i. Thus, if the insides of C_i and C_j overlap, the spirals S_i and S_j intersect. □

Note that the reverse is not true since there exist string graphs that require a high number of intersections between strings and do not correspond to a family of clusterings.



(a) (b)

Figure 4.4.: The transformation from Jordan-curves into strings. This is the same set of clusters as in Fig. 3.2

As mentioned in Section 2.3.3, a graph is not a string graph if it results from a non-planar graph by subdividing every edge. In combination with Lemma 1 this leads to the following corollary:

Corollary 2 *If the cluster-graph G_p of $\mathcal{F}$ is a graph that results from a non-planar graph by subdividing every edge, $\mathcal{F}$ is not strongly embeddable.*

Since k-clique-graphs with $k > 2$ can not have this property, the corollary is only relevant for families of two clusterings. Additionally, the clusters of one family are only allowed to share vertices with a maximum of two different clusters from the other family. Despite these restrictions, Corollary 2 allows us to prove that there exists no planar embedding for the clusterings in Fig. 4.3: since the cluster-graph is a K_5 with subdivided edges and the K_5 is non-planar, there exists no strong embedding for the family of clusters shown in Fig. 4.3.

For the rest of this section, and also in the following sections, we are going to focus on families of two clusterings. This limitation allows us to characterize embeddings by their faces, draw a connection between strong embeddability and vertex planarity and eventually prove the complexity of the decision problem whether two clusterings have a strong embedding or not. It also seems that for practical purposes, visualizations of more than two clusterings at a time are difficult to interpret. Even simple examples like the one in Fig. 4.4(a) show that it is hard for the human perception to distinguish between more than two intersecting curves.

From now on, let $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ be a family of two clusterings on the set of vertices V. In the following we will show that some clusters are not affecting the strong embeddability.

Definition 11 (Singletons) *We call a cluster $C_a \in \mathcal{C}_i$ a* singleton *if it is a subset of a cluster in the other clustering $\mathcal{C}_j$:*

$$C_a \in \mathcal{C}_i \text{ singleton} :\Leftrightarrow \exists C_b \in \mathcal{C}_j, i \neq j : C_a \subseteq C_b$$

In the cluster-graph $G_\mathcal{F}$ a singleton corresponds to a node v of degree $d(v) = 1$. We call such a node a *leaf*. If we remove a leaf v and its adjacent edge e from the cluster-graph, we might create a new leaf: a node that was adjacent to e and one other edge. If we sucessively remove all leaves in the cluster-graph, we end up with its 2-core. This leads to the following definition:

Definition 12 (Reduced Families of Clusters) *We call $\mathcal{F}$* reduced *if its cluster-graph has no leaves. If $\mathcal{F}$ is not reduced, we can* reduce *it by successively removing all singletons and the vertices they contain. We call the resulting family $\mathcal{F}_R$ the* reduction *of $\mathcal{F}$.*

Thus, a family of clusterings $\mathcal{F}$ is reduced if its cluster-graph $G_\mathcal{F}$ corresponds to its proper 2-core. With the example in Fig. 4.5 we show the reduction process of a family of two clusterings: In the first step we can remove the clusters $C_{1,1}$, $C_{2,4}$ and $C_{1,6}$. Now we can remove $C_{1,5}$. This makes $C_{2,3}$ a singleton that we can also remove. In a last step we can remove $C_{1,4}$. Now there are no more singletons in the family of clusterings, and the remaining clusters form a reduced family of clusterings. Note that the remaining cluster-graph is the 2-core of the old cluster-graph.

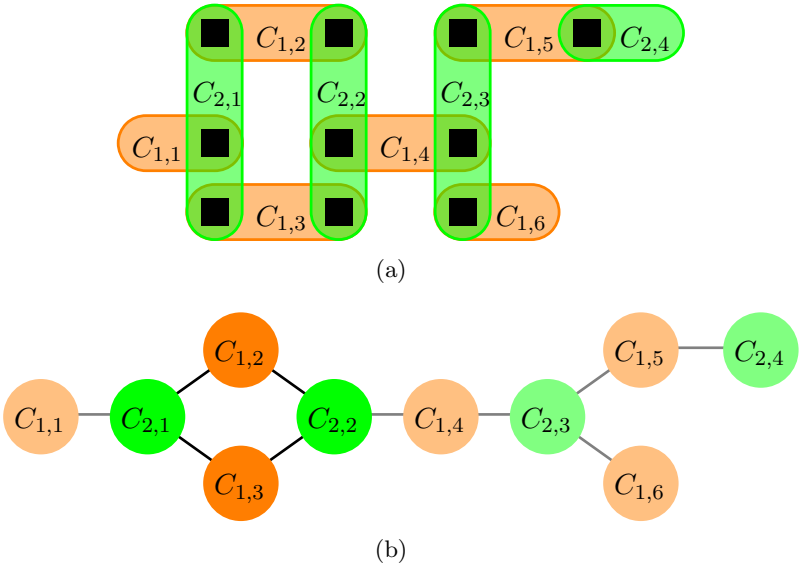


Figure 4.5.: A family of clusterings and the corresponding cluster-graph. After the reduction only the clusters $C_{1,2}$, $C_{1,3}$, $C_{2,1}$ and $C_{2,2}$ will remain.

We will now show that it suffices to examine the reduction of a family of clusterings if we want to test it for strong embeddability. From now on let $\mathcal{F}_R$ be the reduction of $\mathcal{F}$.

Lemma 2 *$\mathcal{F}$ is strongly embeddable if and only if $\mathcal{F}_R$ is strongly embeddable.*

In order to prove the Lemma, we first have to prove the following proposition:

Proposition 2 *In a reduced strong embedding $\mathcal{F}_R$, every cluster is incident to at least one face.*

Proof

Since $\mathcal{F}_R$ is reduced, there exists no cluster that is completely contained in another cluster. This also means that every cluster C_i shares vertices with at least two clusters C_j, C_k from the other clustering. Because C_j and C_k are in the same clustering, the insides of their Jordan curves $J(C_j)$ and $J(C_k)$ are disjoint. Thus, the curve $J(C_i)$ that intersects both $J(C_j)$ and $J(C_k)$ has to cross the region on the outside of $J(C_j)$ and $J(C_k)$ which is the external region $ext(\mathcal{J}(\mathcal{F}_R))$. □

Proof of Lemma 2 (by construction)

If $\mathcal{F}$ has a strong embedding Γ, we can create an embedding Γ_R of $\mathcal{F}_R$ by successively removing Jordan curves that belong to the singletons in $\mathcal{J}(\mathcal{F})$. We can now just remove the vertices that were contained in the singletons and keep the rest of the embedding. The embedding Γ_R is still a strong embedding since we did not create any new intersections between Jordan curves.

Let now Γ_R be a strong embedding of $\mathcal{F}_R$. Let without loss of generality $\mathcal{C}_i$ be the clustering from which the last cluster has been removed in the reduction process from $\mathcal{F}$ to $\mathcal{F}_R$. Let further be $C_a \in \mathcal{C}_i$ the removed cluster and $C_b \in \mathcal{C}_j$ the cluster that contained C_a.

According to Proposition 2, C_b is incident to a face $f \in F(\Gamma_R)$. We draw a Jordan curve $J(C_a)$ whose inside is intersected by $J(C_b)$ in the area where $J(C_b)$ is incident to f. Thus, $J(C_a)$ lies both in f and in the inside of C_b. We place the common vertex of C_a and C_b into the region formed by the intersection of $int(J(C_a))$ and $int(J(C_b))$.

Since $J(C_a)$ is partly placed in f, the embedding upholds the property that every curve is incident to a face. The placement of the vertices in the area of intersections further guarantees that the embedding is still strong. We can thus continue to add curves and their contained vertices for the other clusters that have been removed in the same way. As long as we proceed in the reverse order of their removal, we can go on until we have constructed a strong embedding Γ of $\mathcal{F}$. □

According to Lemma 2, we can add or remove singletons in any family of two clusterings without affecting the strong embeddability. The lemma allows us to focus on reduced strong embeddings in the following sections without loss of generality. In the next section we will examine the relationship between strong embeddability and vertex planarity.

4.2.2. Strong Embeddability and Vertex Planarity

In this section, we will show that strong embeddability of a family of two clusterings $\mathcal{F}$ is equivalent to the vertex planarity of the corresponding hypergraph $H_{\mathcal{F}}$. We will use the equivalence of $\mathcal{F}$ to its reduction $\mathcal{F}_R$ showed in Lemma 2 as a starting point for our proof. Table 4.1 illustrates the chain of equivalences that lead to our proof.

We already proved Lemma 2. Before we prove the remaining two lemmata, we introduce two concepts that facilitate the description of strong embeddings for two clusterings.

The first concept – proper strong embeddings – standardizes the intersections between the Jordan curves. The second concept – condensed strong embeddings – simplifies the treatment of equivalent vertices.

$H_{\mathcal{F}}$ vertex planar	**Theorem 4** $\Leftrightarrow$	$\mathcal{F}$ strongly embeddable
Lemma 3 $\Updownarrow$		$\Updownarrow$ Lemma 2
$H_{\mathcal{F}_R}$ vertex planar	$\Leftrightarrow$ Lemma 4	$\mathcal{F}_R$ on V_R strongly embeddable

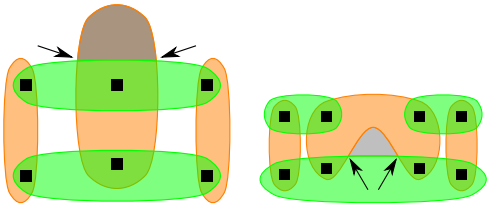
Table 4.1.: The equivalence relationships leading to the proof of Theorem 4

The definition of a strong embedding still gives us a lot of freedom in the choice of the Jordan curves. We will therefore establish a concept to simplify embeddings that makes them more accessible for human interpretation and formal description. We start by defining unnecessary crossings of Jordan curves:

Definition 13 (Unnecessary Crossings) *Two Jordan curves $J(C_a)$ and $J(C_b)$ in a strong embedding $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ of $\mathcal{F}$ have an* unnecessary crossing *if there exists a connected component R_i of a region $R \in \mathcal{R}(\mathcal{J})$ in the embedding that is only bounded by the two curves and that does not contain a vertex.*

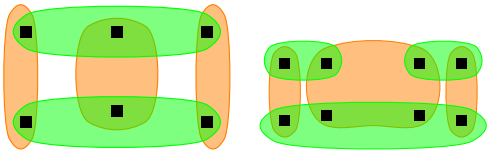
There exist two types of unnecessary crossings: The first type induces a component of a region that lies in the inside of one curve and in the outside of the other (See Fig. 4.6(a)). The second type induces a face of the embedding that is bounded only by the curves of the clusters (See Fig. 4.6(b)). Note that a third case – the component being in the inside of both curves – can not occur since – due to the definition of a strong embedding – this component has to contain a vertex.

Definition 14 (Proper Strong Embeddings) *A strong embedding Γ is* proper *if its Jordan curves do not induce any unnecessary crossings.*



(a) (b)

Figure 4.6.: The two types of unnecessary crossings



(a) (b)

Figure 4.7.: How unnecessary crossings can be avoided in a proper strong embedding

It is easy to see that the properness-property is no real restriction: Unnecessary crossings can be avoided by changing the shape of the Jordan curves. Figure 4.7 shows how both types of unnecessary crossings can be avoided. This proves the following proposition:

Proposition 3 *We can transform every strong embedding into a proper strong embedding.*

Now that we have simplified the curves of strong embeddings, we are going to introduce the second simplification that allows us to only look at one representing vertex for each connected component of a region:

Definition 15 (Condensed Strong Embeddings) *A strong embedding* $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ *is* condensed *if every connected component of a region* $R \in \mathcal{R}(\mathcal{J}(\mathcal{F}))$ *contains at most one vertex.*

Note that a condensed embedding does not necessarily correspond to a condensed corresponding hypergraph. The condensation of a hypergraph only contains one vertex for every equivalence class. In the case of embeddings, it is possible that the intersection between two curves has more than one connected component, requiring equivalent vertices to be placed in each connected component. In Section 4.2.5, we will show that there are hypergraphs that lose the property of strong embeddability if they are condensed.

If a strong embedding is not condensed, we can condense it by choosing one representing vertex for each component of a region and by associating equivalent vertices in the same component region with it. Such an embedding can be expanded again if necessary. The equivalent vertices within the connected components also do not have an influence on the existence of a planar support: we can just add or remove such a vertex v in any planar support by adding an edge $\{u, v\}$ to the representing vertex u or contracting an edge $\{u, v\}$ to remove v.

From now on we will – without loss of generality – assume that every embedding Γ is proper and condensed.

Now we can move on to the next step to the proof of Theorem 4 and show that the vertex planarity of the corresponding hypergraph of a clustering is not affected by the process of reduction.

Lemma 3 *The corresponding hypergraph* $H_{\mathcal{F}}$ *of* $\mathcal{F}$ *is vertex planar if and only if the corresponding hypergraph* $H_{\mathcal{F}_R}$ *of the reduction* $\mathcal{F}_R$ *is vertex planar.*

Proof

If $H_{\mathcal{F}}$ is vertex planar, it has a planar support $G_p(H_{\mathcal{F}})$. We can create a planar support $G_p(H_{\mathcal{F}_R})$ for $H_{\mathcal{F}_R}$ as follows: We successively select a vertex v that is contained in a singleton and contract one of its incident edges $\{u, v\}$, associating the resulting vertex with the vertex u. This does not destroy the planarity and removes the vertices and edges that belong to singletons from the support. We thus end up with a support for $H_{\mathcal{F}_R}$, showing that $H_{\mathcal{F}_R}$ is still vertex planar.

If $H_{\mathcal{F}_R}$ is vertex planar, it has a planar support $G_p(H_{\mathcal{F}_R})$. We now add the nodes of the singletons to $G_p(H_{\mathcal{F}_R})$, connecting them to an arbitrary node v of their containing cluster. We proceed like this until we have a support $G_p(H_{\mathcal{F}})$ for $H_{\mathcal{F}}$. The graph $G_p(H_{\mathcal{F}})$ is planar since we only added trees to $G_p(H_{\mathcal{F}_R})$ that are connected at their root v. These trees can be embedded in any face that is incident to v without creating any crossings. Thus, $H_{\mathcal{F}}$ is vertex planar. □

Now we will show the last of the three lemmata that lead to the proof of the central theorem of this section:

Lemma 4 *A reduced family of two strict partitioning clusterings* $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ *is strongly embeddable if and only if its corresponding hypergraph* $H_{\mathcal{F}}$ *is vertex planar.*

Proof (by construction)

Let $H_{\mathcal{F}}$ be vertex planar and $G_p = (V, E_p)$ a minimal planar support for $H_{\mathcal{F}}$. We take a planar embedding $\Pi(G_p)$ of $G_p = (V, E_p)$ and denote by d the minimal distance between two curves of the embedding outside of the immediate neighborhood of an incident vertex. The positions $P(V)$ of the vertices are the positions of the nodes in $\Pi(G_p)$. For an $\epsilon < \frac{1}{2}d$ we create an enclosing Jordan curve $J(C_i)$ with a distance of $\frac{1}{2}d - \epsilon$ around each subgraph (tree) that is induced by a cluster C_i. All Jordan curves have a minimal distance of 2ϵ from each other except in areas around a vertex where they intersect. Since therefore every area where curves intersect contains a vertex, we have created a strong embedding $\Gamma(G_p) = (\mathcal{J}(\mathcal{F}), P(V))$.

In order to show the other direction, let $\Gamma' = (\mathcal{J}(\mathcal{F}'), P(V'))$ be a strong embedding of $\mathcal{F}$. Without loss of generality, let Γ' be proper and condensed. Thus, every intersection of the insides of two curves contains exactly one vertex, and every vertex is located in a connected component of a region that is bounded by at least two Jordan curves.

Starting at an arbitrary point, we walk around every Jordan curve $J(C) \in \mathcal{J}(\mathcal{F})$. Every time we encounter a crossing with another Jordan curve $J(C')$, we connect the vertex in the intersection of the insides of $J(C)$ and $J(C')$ with the vertex from the last intersection we encountered. Since $\mathcal{F}$ is reduced and condensed, all vertices are covered by this procedure. The result is a planar multigraph $G_m(\Gamma)$ with an embedding $\Pi(G_m(\Gamma))$. We generate a graph $G_p(\Gamma)$ by removing the multiedges of $G_m(\Gamma)$. The graph G_p is a support of $\mathcal{F}$ since – due to the construction – the subgraph induced by every cluster is connected. □

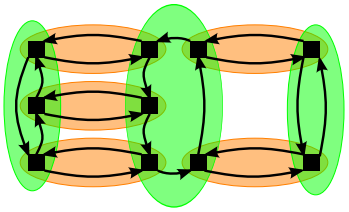


Figure 4.8.: The creation of an induced support from a reduced strong embedding

Now we have all the tools to prove the following main theorem on the relationship between strong embeddability and vertex planarity.

Theorem 4 *A family of two strict partitioning clusterings $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ is strongly embeddable if and only if its corresponding hypergraph $H_{\mathcal{F}}$ is vertex planar.*

Proof

Let the family of clusterings $\mathcal{F}_R$ on V_R be the reduction of $\mathcal{F}$, and $H_{\mathcal{F}_R}$ the corresponding hypergraph of $\mathcal{F}_R$. We will now use the lemmata we introduced in this section to prove the theorem:

The hypergraph $H_{\mathcal{F}}$ is vertex planar if and only if $H_{\mathcal{F}_R}$ is vertex planar (Lemma 3). The hypergraph $H_{\mathcal{F}_R}$ is vertex planar if and only if $\mathcal{F}_R$ on V_R is strongly embeddable (Lemma 4) and $\mathcal{F}_R$ on V_R is strongly embeddable if and only if $\mathcal{F}$ is strongly embeddable (Lemma 2). This proves the theorem. □

The diagram in Table 4.1 we showed at the beginning of this section illustrates how the equivalence relations from the lemmata lead to the equivalence in the theorem.

We are going to reuse the technique from Lemma 4 for generating supports from embeddings and vice versa. We therefore make the following definition:

Definition 16 (Induced Strong Embeddings and Supports) *An embedding $\Gamma(G_p)$ of $\mathcal{F}$ is* induced *by a minimal planar support G_p of $\mathcal{F}$ if it was generated from an embedding $\Pi(G_p)$ in the way described in the proof of Lemma 4.*

A support $G_p(\Gamma)$ of a reduced family of two clusterings $\mathcal{F}$ is induced *by a proper strong embedding Γ of $\mathcal{F}$ if it was created as described in the proof of Lemma 4.*

As stated in Section 2.3.3, it is $\mathcal{NP}$-complete to decide if an arbitrary hypergraph is vertex planar or not. In Section 4.2.4 we will show that this also holds for corresponding hypergraphs of clusterings and thus for the test whether a family of clusterings is strongly embeddable. Before we can prove this, we will examine combinatorial and unique strong embeddings in the following section.

4.2.3. On Combinatorial and Unique Strong Embeddings

The characteristics of embeddings of two clusterings, especially the fact that their corresponding hypergraph is 2-regular, allow us to develop a formal description of embeddings. In this section we will introduce a concept that is similar to the combinatorial embeddings of graphs with the difference that faces are not bounded by edges but by hyperedges that correspond to clusters. This tool will help us to define embeddings that are unique in a sense that any two embeddings can be transformed into each other by a homeomorphic transformation of the Jordan curves.

In the following let $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ be a family of two reduced strict partitioning clusterings on a set of vertices V. We can limit our examinations to reduced families of clusterings without loss of generality since according to Lemma 2, this has no influence on the strong embeddability. We start by defining a combinatorial strong embedding by a circular ordering of the clusters around a face:

Definition 17 (Combinatorial Strong Embeddings) *A* combinatorial strong embedding $\Gamma_c = \Phi(F)$ *of $\mathcal{F}$ is given by a set Φ of cyclical orderings $\phi_1 \dots \phi_m$ of clusters that induce faces $f_1 \dots f_m$ by describing the ordering of their adjacent clusters.*

A strong embedding of a reduced family of clusterings has thus exactly one corresponding combinatorial strong embedding, while a combinatorial strong embedding can induce many different strong embeddings with slight variations of the Jordan curves. We call these embeddings equivalent:

Definition 18 (Equivalent Strong Embeddings) *Two proper strong embeddings Γ_a and Γ_b of $\mathcal{F}$ are* equivalent *if the cyclical ordering of the Jordan curves around the faces induces the same corresponding combinatorial strong embedding.*

Now that we have established classes of equivalent embeddings, the question arises if there exist families of clusterings that have a unique strong embedding.

Definition 19 (Unique Strong Embeddings) *The family of clusterings $\mathcal{F}$ has a* unique strong embedding *if all proper strong embeddings of $\mathcal{F}$ are equivalent.*

Before we can prove that families of clusterings with unique strong embeddings actually exist, we need some tools to facilitate the argumentation in the proof. We will now show that an induced support of an embedding can always induce an embedding that is equivalent to the original one.

Lemma 5 *Let Γ be a proper, condensed embedding of the reduced family of clusterings $\mathcal{F}$ and $G'_p(\Gamma)$ a minimalization of the support $G_p(\Gamma)$ that is induced by Γ. There exists an embedding $\Gamma(G'_p(\Gamma))$ that is induced by $G'_p(\Gamma)$ and that is equivalent to Γ.*

Proof

The faces of the embedding Π of $G_p(\Gamma)$ that resulted in the creation of $G_p(\Gamma)$ are either faces induced by a circle in a cluster or faces that have a corresponding face in Γ (see Fig. 4.8). When we successively remove edges to generate a minimalization of $G_p(\Gamma)$, the circles in the clusters are removed by removing an edge. The remaining minimal support $G'_p(\Gamma)$ (we keep the same embedding) has exactly one face for every face of Γ. When we create an embedding of the clusterings that is induced by the support $\Gamma(G'_p(\Gamma))$ while keeping its embedding, the faces remain the same and are incident to the enclosing Jordan curves of the same clusters. □

From this lemma we can deduce the following corollary:

Corollary 3 *For every proper condensed embedding Γ of $\mathcal{F}$ there exists a minimal support G_p that induces an embedding $\Gamma(G_p)$ that is equivalent to Γ.*

This means that every possible combinatorial strong embedding of $\mathcal{F}$ is induced by a support of $\mathcal{F}$. Since $\mathcal{F}$ is strongly embeddable if and only if it has a planar support, we can conclude that:

Corollary 4 *$\mathcal{F}$ has a unique strong embedding if and only if all strong embeddings $\Gamma(G_p)$ induced by every minimal planar support G_p of $\mathcal{F}$ are equivalent.*

Now we can construct families of clusterings that have a unique strong embedding. We start with a basic construction that we can later flexibly extend when it comes to proving a complexity result and giving counterexamples.

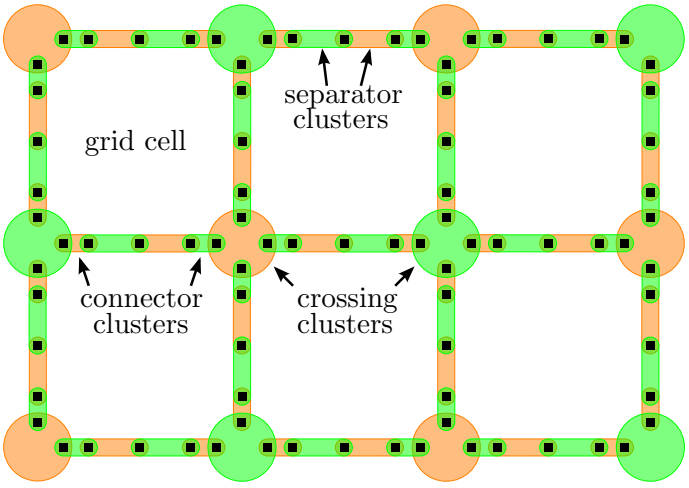


Figure 4.9.: A grid-shaped family of clusterings $\mathcal{F}_{\#}$

Definition 20 (Grid-Shaped Families of Clusterings) *A reduced family of two clusterings $\mathcal{F}_{\#}$ is* grid-shaped *if both clusterings contain only three types of clusters:* crossing clusters*,* separator clusters*, and* connector clusters*. A separator cluster and a connector*

cluster contain exactly two vertices. A connector cluster shares one vertex with a crossing cluster and one with a separator cluster. A separator cluster shares one vertex with a connector cluster and one vertex with another separator cluster. A crossing cluster contains either two, three or four vertices that it shares with connector clusters from the other clustering.

Thus, in a grid-shaped family of clusterings, crossing clusters are connected with each other by a chain of four clusters: two connector clusters on the outside of the chain and two separator clusters in the middle.

Since only clusters from different clusterings can have a common vertex, the clusters from both clusterings alternate and every chain contains a separator cluster from each clustering.

Figure 4.9 is an example of a grid-shaped family of clusterings. Note that in this figure, as well as in the examples that follow, the grid we use is of rectangular structure. The definition would also allow other shapes of grids, but for reasons of clarity we use this special case of a grid structure. To describe the structure of the grid, we introduce a corresponding grid graph:

Definition 21 (Corresponding Grid Graph) *The* corresponding grid graph $G_{\#}(\mathcal{F}_{\#})$ *of a grid-shaped family of clusterings $\mathcal{F}_{\#}$ has a corresponding node for each crossing cluster of $\mathcal{F}_{\#}$ and an edge between two nodes if the corresponding crossing clusters are connected by a chain of separator and connector clusters.*

With this tool we can now make the following statement about grid-shaped families of clusterings.

Lemma 6 *A grid-shaped family of clusterings $\mathcal{F}_{\#}$ has a unique strong embedding if its corresponding grid graph is a subdivision of a 3-connected planar graph.*

Proof

We prove the lemma by showing that every minimal support G_p of $\mathcal{F}_{\#}$ has a unique strong embedding $\Pi(G_p)$ and that the embeddings $\Gamma(G_p)$ of the clusterings that are induced by the supports are all equivalent.

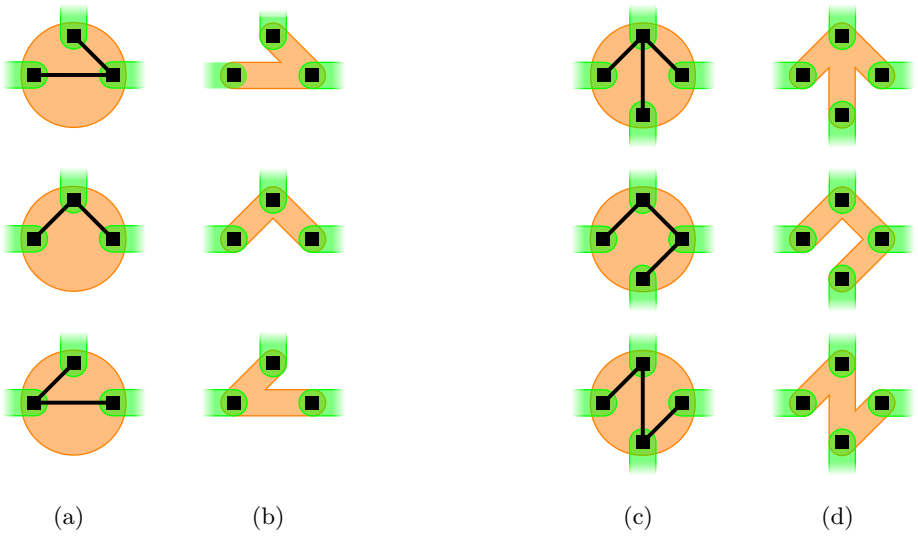
The separator clusters, containing only two vertices, have a unique support: an edge between the two. The same holds for the connector clusters. A chain of separating and connector clusters can thus be treated as a subdivided edge in any support.

A crossing cluster with three vertices allows only the three possible minimal supports shown in Fig. 4.10(a). Figure 4.10(b) shows that the embeddings induced by all three possibilities are equivalent (all faces are incident to the connector cluster).

The same holds for a crossing clusters with four vertices. All possible supports are either a rotation or a reflection of one of the supports shown in Fig. 4.10(c). And all supports induce equivalent embeddings (Fig. 4.10(d)).

Since the corresponding grid graph is a subdivision of a 3-connected planar graph, all possible supports are also subdivisions of 3-connected graphs. Thus they have a unique strong embedding, and since the embedding of every support induces an embedding of $\mathcal{F}_{\#}$ that is equivalent to all other embeddings, $\mathcal{F}_{\#}$ has a unique strong embedding. □

Now that we have constructed families of clusterings with unique strong embeddings, we can extend our grid-shaped families to more interesting constructions. These allow us



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

Figure 4.10.: The different possibilities for a support of a crossing cluster of three or four clusters in $\mathcal{F}_\#$ and the induced clusterings

to construct gadgets for our upcoming proof of the hardness of strong embeddability. We extend the grid by laying additional clusters on top of the grid. The construction is inspired by the "Noodle-Forcing Lemma" of a recent paper by Chaplick et al. on "Bend-Bounded Path Intersection Graphs" [CJKV12]. The authors also use the concept of fixing certain structures ("sausages, noodles, and waffles") on a "grill" to give them a fixed embedding. Yet they deal with string graphs and construct their grid differently than we do.

Before we can properly define such grid-augmenting families of clusterings, we give the following definition of binary separator trees. It helps us to determine clusters in the grid with which the additional clusters can share nodes with.

Definition 22 (Binary Separator Tree) *A* binary separator tree *is a subset of the separator clusters of one clustering of a grid-shaped family of clusterings $\mathcal{F}_\# = \{\mathcal{C}_1, \mathcal{C}_2\}$ that we can construct as follows:*

Let $G^*_\#(\mathcal{F}_\#)$ be the dual graph to the corresponding grid graph of $\mathcal{F}_\#$ and T^* be a subgraph of $G^*_\#(\mathcal{F}_\#)$ that is a binary tree. Let $T \subset \mathcal{C}_1 \cup \mathcal{C}_2$ be the set of clusters that induce the dual edges of T^* in the construction of the corresponding grid graph. A subset of T that consists of all separator clusters of one of the clusterings is called binary separator tree*.*

With its set of separating clusters, a binary separator tree basically describes a binary tree of neighboring grid cells in a grid-shaped family of clusterings. With this concept, we can define a very versatile class of families of clusterings that are augmentations of grid-shaped families and that still have unique strong embeddings.

Definition 23 *A* grid-augmenting *family $\mathcal{F}$ of two clusterings on a set V is a grid-shaped family $\mathcal{F}_\#$ of two clusterings on a set V' with a unique embedding that has been extended by some additional vertices and clusters containig these vertices. We call these additional clusters* augmenting clusters*.*

Every augmenting cluster C is only allowed to share vertices with separator clusters of the other clustering. These separator clusters have to be placed around a grid cell in one of the ways shown in Fig. 4.11:

- *C enters and leaves a cell by sharing a vertex with two of its separator clusters (See Fig. 4.11(a) and 4.11(b))*
- *C branches in a cell by sharing a vertex with three of its separator clusters (See Fig. 4.11(c))*
- *C crosses another augmenting cluster C' in a cell by sharing a vertex with two separator clusters on opposite sides of the cell – diagonally to C' – and by sharing a common vertex with C' for each cell in which they cross (See Fig. 4.11(d))*
- *C ends at a cell by sharing just one vertex with a separator cluster of that cell (See Fig. 4.11(e)). Multiple (mutually distinct) clusters can end at the same cell.*

Additionally, all separator clusters that share nodes with an augmenting cluster have to form a binary separator tree in $\mathcal{F}_{\#}$.

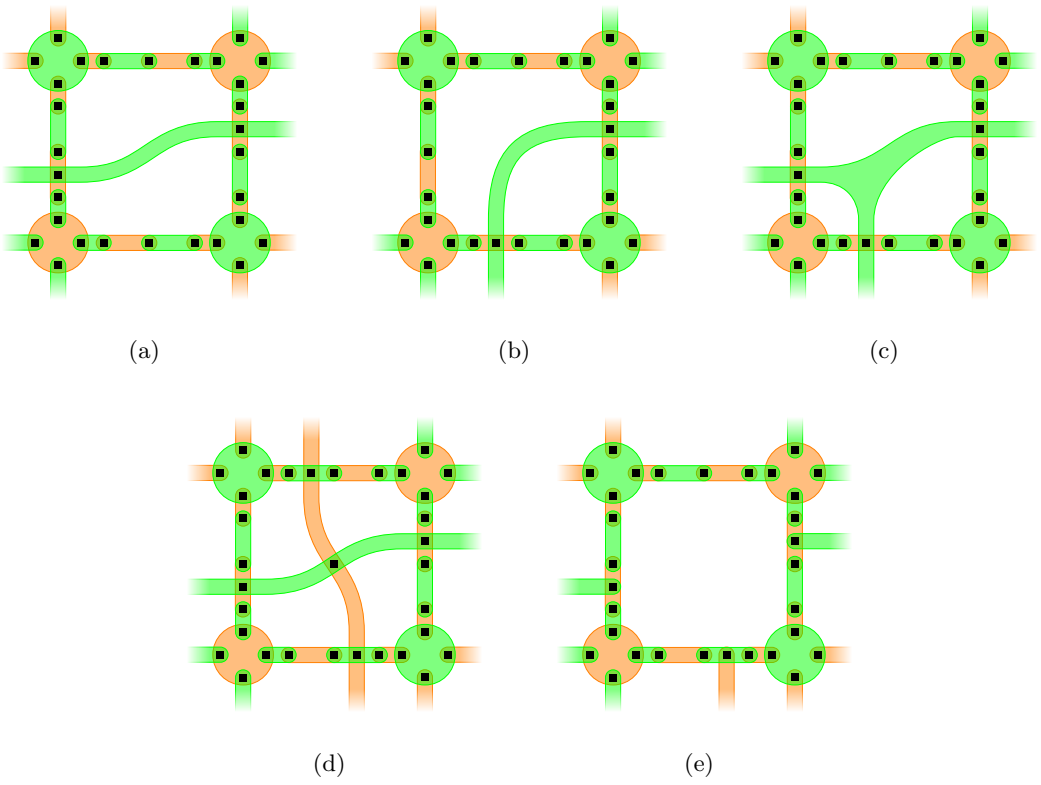


Figure 4.11.: The different possibilities for clusters in a grid-augmenting family of two clusters $\mathcal{F}$

We will now prove the following theorem:

Theorem 5 *Every grid-augmenting family of two clusterings $\mathcal{F}$ has a unique strong embedding.*

Proof

In the proof we will go over all possibilities how a cluster can be embedded in a grid cell shown in Fig. 4.11. According to Lemma 6, every support of a grid-shaped family of clusterings whose corresponding grid graph is 3-connected has a unique strong embedding.

The embedding of the support of the underlying grid does not change in the augmentation since we only add edges (or subdivisions if a node is added to a separator cluster) to a graph with a unique embedding. We will show that every minimal planar support G_p of the grid augmentation $\mathcal{F}$ still has a unique strong embedding and that the induced embeddings of every support are equivalent.

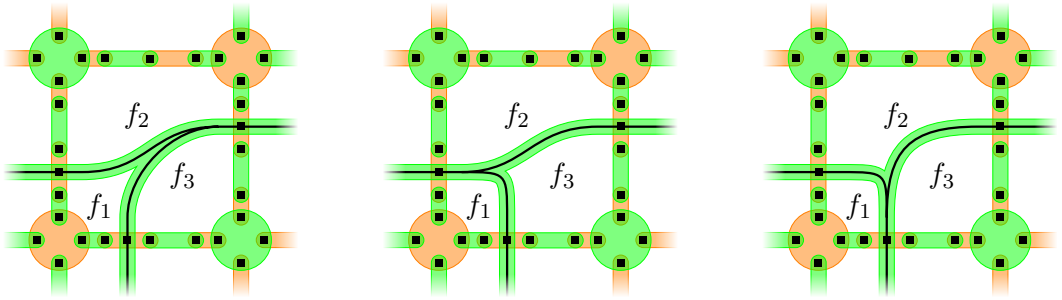
When adding a new augmenting cluster C_a that shares a common vertex with a separator cluster C_s, there are two main possibilities: either the cluster ends in this vertex, i.e., it only shares common vertices with the separator clusters of one of the adjacent cells of C_s (Fig. 4.11(e)), or the cluster passes through the separator cluster, sharing one or more common vertices with both adjacent cells of C_s.

In the first case, C_s can be regarded as equivalent to a crossing cluster that shares vertices with three other clusters (Fig. 4.10(b)). We have shown that in this case, all possible minimal supports are locally 3-connected and induce equivalent embeddings of the clusterings.

In the other case, we have the augmenting cluster sharing vertices with other separator clusters in the two neighboring cells (e.g. Fig. 4.11(a) and 4.11(b)). In this case, due to fixed embedding of the grid, the only option for a planar support is to have a path in the separator cluster and a path in the augmenting cluster which are crossing in the common vertex of the augmenting cluster with the separator cluster.

When two augmenting clusters cross in a cell (Fig. 4.11(d)), there is also just one way to create a support: adding an edge from the vertex in the middle to the vertices in the separating clusters. Any other support would violate planarity. Especially a placement of the common vertex of the two augmenting clusters into another grid cell would cause one of the clusters to induce a support that is disconnected.

When a cluster shares a vertex with three separating clusters of a cell (Fig. 4.11(c)), there are three different ways to create a minimal planar support. Figure 4.12 shows that no matter which embedding we choose, the faces of the corresponding embeddings of $\mathcal{F}$ are bounded by the same clusters in the same order, and thus the embeddings are equivalent.



(a) (b) (c)

Figure 4.12.: The different possibilities for a support if a cluster shares vertices with three separating clusters of a cell

Due to the property of the binary separator tree, the induced subgraph of the augmenting clusters does not contain a circle. Thus, there is no possible choice for an edge to be removed. Since in all of the cases we discussed, we only added edges to a support that is already a subdivision of a 3-connected graph, all possible supports of $\mathcal{F}$ remain subdivisions of 3-connected graphs and therefore have fixed embeddings. Furthermore, we showed that

all possible induced embeddings of $\mathcal{F}$ are equivalent and hence that $\mathcal{F}$ has a unique strong embedding. □

We will also establish a schematized method of drawing grid-augmenting families of clusters. Since the grid only serves as a helping structure, we do not draw every single cluster but only the general structure of the corresponding grid graph. See Fig. 4.13 for an example.

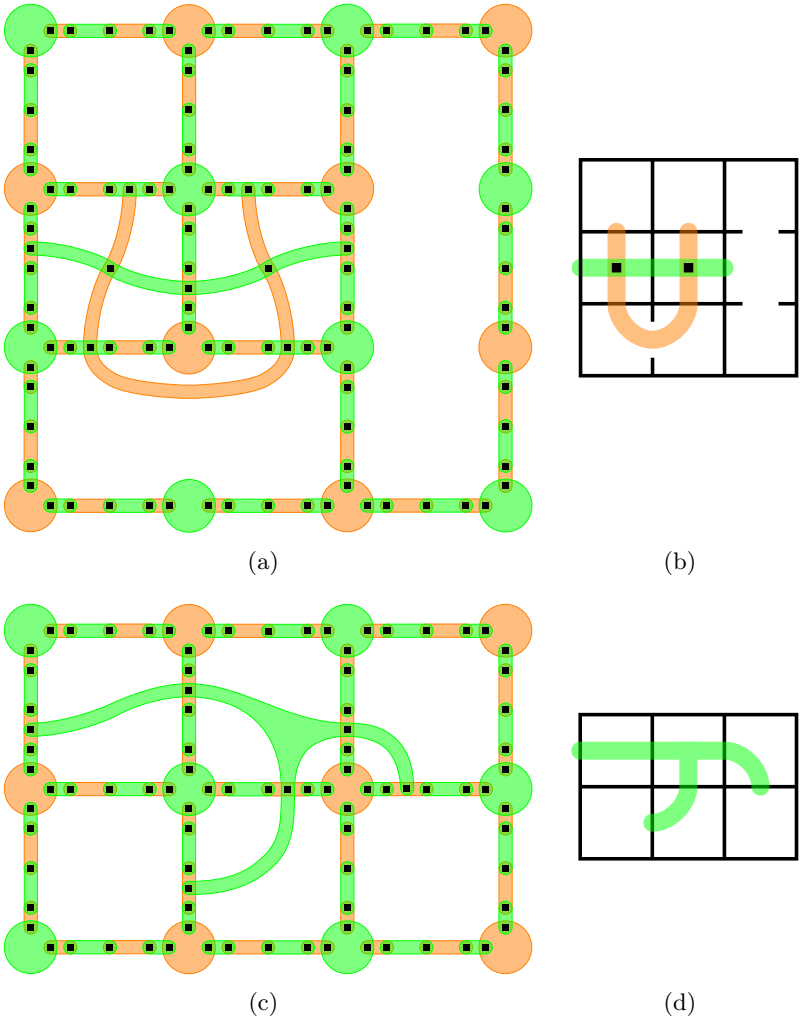


Figure 4.13.: Two grid-augmenting families of clusters and their schematized drawings

4.2.4. Complexity of the Test for Strong Embeddability

As stated in Chapter 2, it is an $\mathcal{NP}$-complete problem to decide if an arbitrary hypergraph is vertex planar. In this section we will show that this also holds for the special kind of hypergraphs that are induced by a family of two clusterings. We define our decision problem as follows:

Problem 1 (k-CSE) *The decision problem* k-Clusterings-Strong-Embeddability *(k-CSE) is the following: Is a family $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ of k clusterings strongly embeddable or not.*

In this section, we will prove the $\mathcal{NP}$-completeness of k-CSE for the case of two clusterings ($k = 2$). This implies the $\mathcal{NP}$-completeness for $k > 2$ since an instance of 2-CSE can be

transformed into an instance of k-CSE by simply adding $k-2$ trivial clusterings that contain only one cluster comprising all vertices.

Additionally, due to the equivalence of vertex planarity to strong embeddability for two clusterings, our results show the $\mathcal{NP}$-completeness of the check for vertex planarity in 2-regular hypergraphs. Since we can extend a 2-regular hypergraph to a k-regular hypergraph by adding $k-2$ trivial hyperedges, the results imply the $\mathcal{NP}$-completeness of the check for vertex planarity in k-regular hypergraphs for $k > 2$.

Our proof has been inspired by the proof by Buchin et al. [BvKM$^+$10], showing the $\mathcal{NP}$-completeness of finding a planar support for 2-outerplanar hypergraphs by reducing it to 3-SAT. Unfortunately, their proof – as well as the original proof for the hardness of vertex planarity by Johnson and Pollak [JP87] – does not work for the corresponding hypergraphs of clusterings that we are examining. Both proofs require certain vertices to be incident to more than two hyperedges. Contrary to that, the following proof also works for this special class of hypergraphs induced by clusterings:

Theorem 6 2-CSE *is $\mathcal{NP}$-complete.*

Proof

It is easy to check if a given embedding $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ of a family of two clusterings $\mathcal{F}$ fulfills the criteria for strong embeddability: Since in this case, strong embeddability is equivalent to the vertex-planarity of the corresponding hypergraph, we just have to check whether a given graph is a planar support of $\mathcal{F}$. Since this can be done in polynomial time, k-CSE is in $\mathcal{NP}$.

To show that the problem is $\mathcal{NP}$-complete, we perform a polynomial reduction from Planar-Monotone-3-Sat. We first briefly review the history of $\mathcal{NP}$-complete SAT-problems leading to our problem:

3-SAT

An instance of 3-Sat consists of a set $\mathcal{U} = \{x_1, \ldots, x_n\}$ of n Boolean variables and a formula $\mathcal{C} = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ in *conjunctive normal form* defined over $\mathcal{U}$. Each clause C_i, $1 < i < m$ is a disjunction of not more than three literals from the variables in $\mathcal{U}$. 3-Sat is one of "Karp's 21 $\mathcal{NP}$-complete problems" [Kar72]. It is thus $\mathcal{NP}$-complete to decide whether there exists an assignment to the variables in $\mathcal{U}$ that fulfills all clauses in $\mathcal{C}$.

MONOTONE-3-SAT

Monotone-3-Sat is a specialization of 3-Sat where all clauses in $\mathcal{C}$ are *monotone*. A clause is monotone if it consists either of only positive or of only negative literals. Garey and Johnson [GJ79] showed that Monotone-3-Sat is $\mathcal{NP}$-complete.

PLANAR-3-SAT

An instance of Planar-3-Sat is an instance of 3-Sat whose *corresponding graph* is planar. The corresponding graph of a formula $\mathcal{C}$ in conjunctive normal form is defined as $\mathcal{G} = (\mathcal{C} \cup \mathcal{U}, \mathcal{E})$ where $\mathcal{E} = \{(x_i, C_j) \mid x_i \in C_j \vee \overline{x_i} \in C_j\}$. Lichtenstein [Lic82] proved that Planar-3-Sat is $\mathcal{NP}$-complete as well. Knuth and Raghunathan [KR92] showed that the problem stays $\mathcal{NP}$-complete if it is limited to problems that have a *rectilinear representation*, i.e., the nodes corresponding to the variables are ordered, aligned horizontally and connected to their neighbors in the ordering. Clauses are located either above or below the line of variables and connected by rectangular edges without crossings. The graph in Fig. 4.14(a) is an example of a rectilinear representation.

PLANAR-MONOTONE-3-SAT

De Berg and Khosravi [dBK09] showed that Planar-3-Sat stays $\mathcal{NP}$-complete if it is restricted to instances with only monotone clauses that have a rectilinear representation where all clauses containing negative literals are embedded on one side of the variables and all clauses containing positive literals are embedded on the other side. This problem is called Planar-Monotone-3-Sat. The example in Fig. 4.14(a) together with the adjoining clauses forms an instance of Planar-Monotone-3-Sat: all clauses above the variables only consist of positive literals, whereas all clauses below the variables only consist of negative literals. Note that the clause C_1 consists only of two distinct variables. The double-occurrence of variables in clauses is not excluded in the definition of Planar-Monotone-3-Sat, and we will take this case into account in our proof.

We will now show how to perform a reduction from Planar-Monotone-3-Sat to 2-CSE in polynomial time.

In our reduction we construct a family of clusterings that has a unique embedding for most of its clusters. The clusters whose embedding is not unique can only cross a certain face of the embedding if no cluster from the other color crosses this face. This face will represent a variable, and the crossing of a cluster of a color represents an assignment of this variable. If and only if there exists a fulfilling assignment of the variables, we can find a strong embedding in which no two clusters intersect without a common vertex. We now show how we can construct such a family of clusterings.

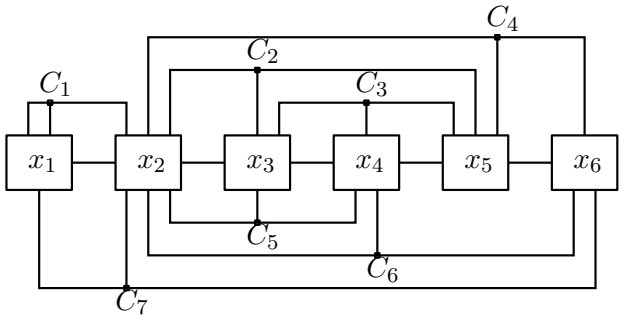
We begin by constructing a grid-augmenting family $\mathcal{F}$ of two clusterings in the way shown in the schematized drawing of Fig. 4.14(b). The grid has an upper and a lower half, each consisting of one row for each cluster. The two halves are separated by another row of larger cells: each cell in this row represents a variable and stretches over c_i columns if the variable x_i is contained in c_i clauses. We will call these cells *variable cells*. Obviously, the corresponding grid graph is 3-connected, and we thus have a unique strong embedding of the grid-shaped family of clusterings.

For each clause C_i we add two augmenting clusters to the grid: a cluster C_i^a in the upper half and a cluster C_i^b in the lower half. The ordering of the clusters in the rows corresponds to the vertical ordering of their edges in the planar rectilinear representation. We use orange clusters for the clauses consisting of negative literals and green clusters for the clauses that consist of positive literals. In the upper half, the orange clusters are placed in the outer rows, in the lower half in the inner rows, close to the variable cells. The clusters have an **E**-shape with their arms ending in the cells of the variables of their corresponding clause. Due to the planarity of the rectilinear representation of the clauses, clusters of the same color can be embedded without intersections in the grid. We add a common vertex in a cell if two clusters of different colors intersect. The green clusters all end in the upper right and lower left, the orange clusters in the upper left and lower right of the variable cells. They keep the same relative ordering among clusters of their own color that is induced by the ordering of the edges in the planar embedding of the instance of Planar-Monotone-3-Sat.

If a variable is contained twice in a clause, we only create one arm ending in the cell of the variable (See cluster C_1^a and C_1^b). Otherwise, the same cluster would have two endpoints in the corresponding variable cell and the embedding would not be unique anymore.

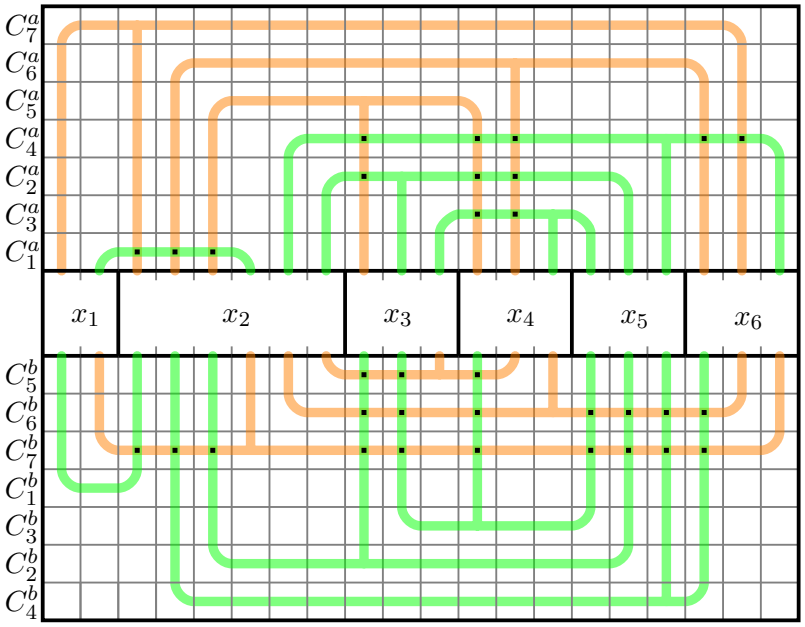
Lemma 6 guarantees that $\mathcal{F}$ has a unique strong embedding, and the two clusterings in $\mathcal{F}$ can be constructed in polynomial time since there are at most $6 \cdot m^2 + m$ grid cells where m is the number of clauses.

We are now going to modify $\mathcal{F}$ to a family of clusterings $\mathcal{F}'$ by combining the two clusters C_i^a and C_i^b of each clause to a single cluster C_i (See Fig. 4.14(c)). This destroys the property

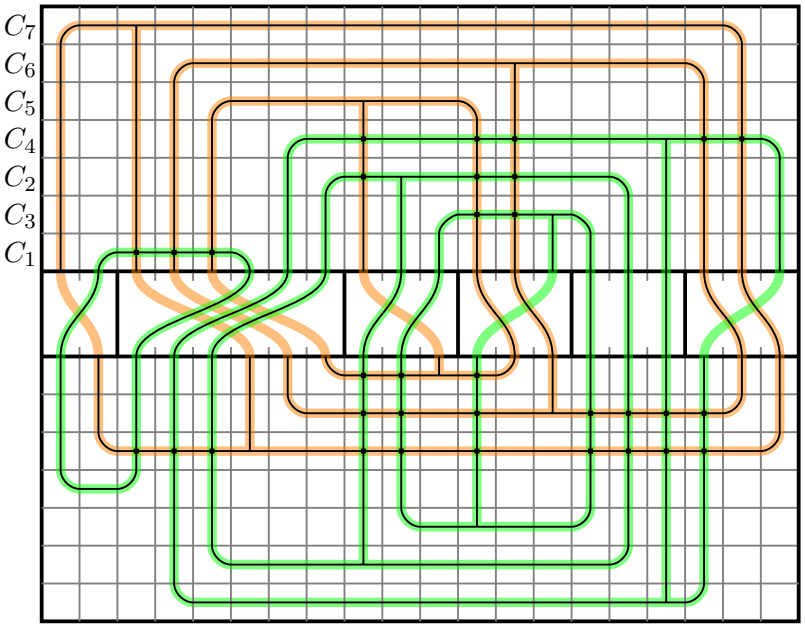


$C_1 = (x_1 \vee x_1 \vee x_2)$
$C_2 = (x_2 \vee x_3 \vee x_5)$
$C_3 = (x_3 \vee x_4 \vee x_5)$
$C_4 = (x_2 \vee x_5 \vee x_6)$
$C_5 = (\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$
$C_6 = (\overline{x_2} \vee \overline{x_4} \vee \overline{x_6})$
$C_7 = (\overline{x_1} \vee \overline{x_2} \vee \overline{x_6})$

(a)



(b)



(c)

Figure 4.14.: The construction of a grid-augmented family $\mathcal{F}$ of two clusterings from an instance of Planar-Monotone-3-Sat and a planar support in the combined case $\mathcal{F}'$

of uniqueness of the embedding. Now the embedding is strong if the induced support for every cluster is a connected tree. Therefore, in a strong embedding a combined cluster is cut at two different places in order to induce a minimal support. These places for the cuts are the only degree of freedom we have in an embedding of $\mathcal{F}'$. Even though the embedding is no longer unique, the combination of the clusters allows us to make the following claim:

The family of clusterings $\mathcal{F}'$ is strongly embeddable if and only if the corresponding instance of Planar-Monotone-3-Sat *is satisfiable.*

If the instance is satisfiable, there exists a satisfying assignment to the variables. We create a support for $\mathcal{F}'$ by adding edges in the variable cells connecting the green clusters if the variable is TRUE and connecting the orange clusters if the variable is FALSE. Since every clause contains at least one literal that evaluates to TRUE, the upper and lower half of the cluster can be connected through the cell of the variable whose literal evaluates to true. No two clusters cross in a variable cell since every variable can be either TRUE or FALSE, i.e., crossed only by green or orange clusters, respectively.

Figure 4.14(c) shows a planar support for the clusterings that corresponds to the following satisfying assignment of the variables: x_1 = TRUE, x_2 = TRUE, x_3 = TRUE, x_4 = FALSE, x_5 = TRUE and x_6 = FALSE. Note that for reasons of clarity, the support in this example is not minimal: a minimal support would not allow circles within the subgraph induced by the augmenting clusters. Yet, any minimalization would obviously still induce a fulfilling assignment of the variables.

The other direction of the proof requires more thought:

If $\mathcal{F}'$ is strongly embeddable, there exists a planar support G_p for $\mathcal{F}'$ that is induced by a strong embedding Γ'. Thus, for every cluster C_i there exists a path in its induced subgraph $G_p[C_i]$ of G_p between any vertex in the upper half and any vertex in the lower half of the grid. Since the grid has a unique strong embedding, the path must cross at least one of the variable cells. It obviously can not leave the variable cell while crossing it since it does not share any vertex with the adjacent clusters of the cell besides the two separator clusters through which it crosses.

We will from now on use the notion of a cluster C_i *crossing* a grid cell g in Γ' if in the embedding of the induced support G_p of Γ an edge of $G_p[C_i]$ would be placed within the corresponding face of the cell. We call a vertex v a *free vertex* if it is shared by two of the augmenting clusters in $\mathcal{F}$. We call a strong embedding Γ^* of $\mathcal{F}'$ a *canonical strong embedding* of $\mathcal{F}'$ if all free vertices are placed in the same grid cell as in the unique strong embedding Γ of $\mathcal{F}$. We call the position of a free vertex in the canonical strong embedding its *original position.* A canonical embedding means in particular that no free vertices are placed inside a variable cell.

In the case of a canonical strong embedding, each variable cell is crossed by clusters of at most one color, and we have found a fulfilling assignment of the variables in the corresponding instance of Planar-Monotone-3-Sat: a variable is set to TRUE if its cell is crossed by green clusters, FALSE if its cell is crossed by orange clusters, and its value is not relevant if its cell is not crossed at all.

However, since the embedding is not unique anymore, we can not guarantee that a planar support induces a canonical strong embedding. It could be the case that there exists a strong embedding in which one or more of the free vertices are actually placed inside of a variable cell. In that case we can not deduce an assignment of the variables anymore.

Luckily, we can show that if there exists a strong embedding Γ' of $\mathcal{F}'$, we can transform it into a canonical strong embedding Γ^*.

We will first show the principal proceeding for *moving* a single free vertex from a variable cell in Γ' back into its original cell as given in the canonical strong embedding. We then show that this method also works if there are multiple vertices in a variable cell.

Let us assume that an orange cluster C_i and a green cluster C_j share a common free vertex v that is placed in the variable cell x_k in the strong embedding Γ'. Since in the canonical strong embedding all common vertices of C_i and C_j were placed in grid cells of the upper or the lower half (see Fig. 4.14(b)), the original position of v is in one of these grid cells. Let this grid cell be g and let, without loss of generality, g be a cell in the upper half of the grid.

Since Γ' is a strong embedding, C_i and C_j do not cross in g since a crossing between C_i and C_j would require the vertex v to be placed in g. Thus, the placement of v in a cell other than g results in either the orange cluster C_i or the green cluster C_j being cut in the cell. Without loss of generality, let the cluster that is cut be C_j. Thus, the green cluster C_j does not cross g anymore.

Since Γ' is a strong embedding, there must be a support G_p despite the cut in which the induced subgraph $G_p[C_j]$ is connected. In particular, there must be a path in $G_p[C_j]$ that connects the two vertices that C_j shares with the separator clusters of g (note that these vertices are not explicitly shown in the schematized drawings). Due to the placement of the augmenting clusters, this path has to pass through the lower half of the grid and back, crossing two variable cells. Thus, there exists another variable cell x_l besides x_k that is crossed by an edge of $G_p[C_j]$. Therefore, C_j does not need the connection through x_k, and we can create another strong embedding, moving v back in g. In this embedding, the green cluster C_j is cut in the variable cell x_k instead of g.

We have just explained how we can move a single vertex from a variable cell in Γ' back to its original position given by the canonical embedding. But – since the free vertices are not independent – how can we guarantee that we can always move all free vertices of a variable cell back to their original positions? We will now show that we can do this successively with all vertices in a variable cell. In order to do that, we introduce the notion of determined color of a variable cell and then show that every cell can have at most one determined color.

If a variable cell x_k in Γ' is crossed by a cluster C_i and C_i does not cross any other variable cell, we call the color of C_i a *determined color* of x_k. We call C_i a cluster that *determines* the color of x_k.

Since a cluster C_i that determines the color of a variable cell x_k does not cross any other variable cell, it can not be the cluster that is cut when one of its contained free vertices is misplaced. Otherwise its induced subgraph $G_p[C_i]$ would be disconnected. Thus C_i is required to cross the variable cell x_k, no matter which free vertices we move. On the other hand, all clusters that are not determining the color can be cut in x_k if we move their contained vertices to their original cell since they have a second connection between their upper and lower halves through another variable cell. If there is a color that is not a determined color of x_k, we can move the contained free vertices of all clusters of that color in x_k back to their original position.

Now we just have to show that a variable cell can never have two different determined colors:

Let us assume that the cell x_k has two different determined colors. This would require the cell x_k being crossed by a determining orange cluster C_i and a determining green cluster C_j. This means that for both C_i and C_j, the only connection between their upper and their lower part is through the cell x_k. Due to their intersection in x_k, the two clusters

also have to share a common free vertex v that is placed in x_k. Let g be the original grid cell of v in which C_i and C_j intersect in the canonical embedding. The removal of v from g requires a cut in one of the two clusters. Since none of the two has another connection between its upper and its lower half, the cut means that one of the subgraphs $G_p[C_i]$ and $G_p[C_j]$ in the support is not connected. This is a contradiction to the initial condition of Γ' being a strong embedding.

Thus, in every strong embedding every variable cell can have at most one determined color. We can therefore pick an arbitrary variable cell and successively move all vertices back to their original positions from the canonical embedding. Note that some of the other cells that were undetermined before might now have a determined color. Yet, as we showed in the example of moving a single vertex, the resulting embedding remains a strong embedding. We can thus perform this operation for all variable cells that contain vertices in an arbitrary ordering and end up with a canonical strong embedding Γ^* which gives us a fulfilling assignment of the variables. Thus, the corresponding instance of Planar-Monotone-3-Sat is satisfiable if and only if there exists a strong embedding of $\mathcal{F}'$. □

As mentioned at the beginning of this section, we can draw the following two corollaries from Theorem 6:

Corollary 5 *k-CSE is $\mathcal{NP}$-complete for $k \geq 2$.*

Corollary 6 *The test for vertex-planarity of a hypergraph remains $\mathcal{NP}$-complete for k-regular hypergraphs with $k \geq 2$.*

In the next two sections, we are going to introduce subclasses of strong embeddability that require additional constraints.

4.2.5. Single-Intersection and Path-Based Strong Embeddability

For the subclasses of strong embeddability in this section, we will go back to an arbitrary number of clusterings: let $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ be an embedding of a family of k strict partitioning clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ on a set V.

The embeddings of the following class require that equivalent vertices are placed in the same region:

Definition 24 (Single-Intersection Strong Embeddings) *An embedding Γ of a family of k clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ is a* single-intersection strong embedding *(SISE) if it is a strong embedding and for every pair of clusters $C_a, C_b \in \bigcup_{\mathcal{C} \in \mathcal{F}} \mathcal{C}$ the intersection between the insides of $J(C_a)$ and $J(C_b)$ is a connected region:*

$$|comp(int(J(C_a)) \cap int(J(C_b)))| = 1$$

We call $\mathcal{F}$ single-intersection strongly embeddable *if there exists a single-intersection strong embedding for $\mathcal{F}$.*

This restriction corresponds to the definition of Euler diagrams (Section 2.3.2) when limiting the connectedness-property to the regions that actually contain a vertex: In an Euler diagram every region has to be connected, whereas in a single-intersection strong embedding only the regions containing a vertex (i.e., regions in the inside of k curves) have to be connected. We can characterize single-intersection strong embeddings by the condensation of the corresponding hypergraph:

Theorem 7 *If $\mathcal{F}$ contains only two clusterings, it is single-intersection strongly embeddable if and only if the condensation $H'_{\mathcal{F}}$ of the corresponding hypergraph $H_{\mathcal{F}} = (V, \mathcal{S})$ is vertex planar.*

Proof

In a single-intersection strong embedding of $\mathcal{F}$ all equivalent vertices are placed in the same connected component of their region. We can thus remove all but one representing vertex for each region and still have a strong embedding of two families of clusterings $\mathcal{F}'$. Their corresponding hypergraph is the condensation $H'_{\mathcal{F}}$ of $H_{\mathcal{F}}$ since any two clusters share at most one vertex, and $H'_{\mathcal{F}}$ is vertex planar since the clusterings have a strong embedding.

Let V be the underlying set of vertices of $\mathcal{F}$. If the condensation $H'_{\mathcal{F}}$ of the corresponding hypergraph $H_{\mathcal{F}}$ is vertex planar, there exists a strong embedding Γ' of a family of clusterings $\mathcal{F}'$ that corresponds to the hypergraph $H'_{\mathcal{F}}$. Let $V' \subseteq V$ be the underlying set of $\mathcal{F}'$. The difference between $\mathcal{F}'$ and $\mathcal{F}$ is only the underlying set; their cluster-graphs are the same. We can therefore just place the additional vertices from V that are not in V' into the regions of their representing vertex in Γ' and end up with a strong embedding Γ of $\mathcal{F}$. □

Now we will show that this class is a true subclass of the strongly embeddable families of clusterings:

Lemma 7 *There exists a family of two clusterings $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ on a set V that is not single-intersection strongly embeddable.*

Proof

We use the counterexample in Fig. 4.13(a). The grid structure enforces a unique strong embedding of the clusterings. In this embedding the two augmenting clusters have two points of intersection that require the two equivalent common vertices of the two clusters. □

It is still an open question whether the test for single-intersection strong embeddability is also $\mathcal{NP}$-complete. Since our proof for strong embeddability in the last section uses multiple intersections between clusterings, we can not apply it to this problem.

The following class of embeddings requires a linear ordering of the vertices in each cluster:

Definition 25 (Path-Based Strong Embeddings) *A family of k strict partitioning clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ is* path-based strongly embeddable *if its corresponding hypergraph $H_{\mathcal{F}} = (V, \mathcal{S})$ has a path-based planar support G_p, i.e., for every hyperedge $S \in \mathcal{S}$ the induced subgraph $G_p[S]$ of G_p is a path.*

We call an embedding that is equivalent to an embedding induced by a path-based planar support a path-based strong embedding *(PBSE).*

Brandes et al. [BCPS11b] already defined the concept of path-based supports for arbitrary hypergraphs. They stated it to be $\mathcal{NP}$-complete by referring to the original proof of the $\mathcal{NP}$-completeness of vertex planarity by Johnson and Pollak [JP87] that uses a path-based support. They found a polynomial-time method to find path-based supports that are trees.

However, as for single-intersection strong embeddability, it is still an open question whether the problem remains $\mathcal{NP}$-complete when limited to the special instances of corresponding hypergraphs of families clusterings.

Yet, we can show the relationship to string-graphs (see Section 2.3.3 for the definitions):

Theorem 8 *A family of two strict partitioning clusterings $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ on a set V is path-based strongly embeddable if and only if its cluster-graph $G_{\mathcal{F}} = (V_{\mathcal{F}}, E_{\mathcal{F}})$ is a string-graph with a representation $R = \{R(v), v \in V_{\mathcal{F}}\}$ in which the number of intersections between two curves $R(v_i), R(v_j) \in R$ that are representing the nodes $v_i, v_j \in V_{\mathcal{F}}$ is less than the number of common vertices of their corresponding clusters C_i, C_j.*

Proof (by construction)

We first show how to construct a path-based strong embedding $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ from the representation R: We place a vertex $v \in C_i \cap C_j$ in every point of intersection between the curves $R(v_i), R(v_j) \in R$. This is possible since the number of intersections of $R(v_i)$ and $R(v_j)$ does not exceed the number of common vertices of C_i and C_j. We then draw an enclosing curve J_i around each curve $R(v_i)$ with a sufficiently small distance, such that two curves J_i and J_j only intersect in the immediate neighborhood around a vertex $v \in C_i \cap C_j$. The result is a condensation of the desired strong embedding. We can now place the remaining vertices in $C_i \cap C_j$ in the intersections between the insides $int(J_i)$ and $int(J_j)$ of the curves J_i and J_j and end up with a strong embedding Γ.

Let now $\Gamma = (\mathcal{J}(\mathcal{F}), P(V))$ be a path-based strong embedding of $\mathcal{F}$. There exists thus a path-based support G_p of $\mathcal{F}$. We are now going to modify G_p to a multigraph G'_p by replacing an edge e by n multiedges if e is contained in the induced path of n clusterings. Obviously, we can still find a planar embedding Π of G'_p. We now take all edges on the path $G_p[C_i]$ that is induced by a cluster C_i and combine them to the simple curve $R(v_i)$. If we use a different multiedge for each induced path, the combination of all curves $R(v_i)$ is a string-representation of the cluster-graph $G_{\mathcal{F}}$ with n crossings between two strings if the corresponding clusters share n vertices. □

We will refer to such string graphs with a limited number of intersections between strings as *individually bounded intersection string graphs.*

Even though it seems that, at least for the case of two clusterings, most families of clusterings are path-based strongly embeddable if they are strongly embeddable, we will show an example to prove the following lemma:

Lemma 8 *There exists a family of two clusterings $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ on a set V that is not path-based strongly embeddable.*

Proof

We use the counterexample in Fig. 4.13(a). Due to the unique strong embedding, the green augmenting cluster can not have a path-based support. All possible supports for the green cluster are trees (see also Fig. 4.12 for the possible supports). □

We also define a class of clusterings that are both single-intersection and path-based strongly embeddable:

Definition 26 (Single-Intersection Path-Based Strong Embeddings) *A family of k strict partitioning clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ on a set V is* single-intersection path-based strongly embeddable *if it is single-intersection strongly embeddable and path-based strongly embeddable. We call an embedding that fulfills both requirements a* single-intersection path-based strong embedding *(SIPBSE).*

From the two theorems on path-based and single-intersection strong embeddings we can draw the following corollary:

Corollary 7 *A family of k strict partitioning clusterings $\mathcal{F} = (\mathcal{C}_i)_{i \leq k}$ is single-intersection path-based strongly embeddable if and only if the cluster-graph is in 1-string.*

Proof

If the cluster-graph $G_{\mathcal{F}}$ is in 1-string, we can generate a single-intersection path-based strong embedding of $\mathcal{F}$ from a string representation of $G_{\mathcal{F}}$ by drawing an enclosing curve around each string as shown in the proof of Theorem 8. Since there exists only one point of intersection between any two strings, the path-based strong embedding fulfills the criteria of a single-intersection path-based strong embedding.

If we have a single-intersection path-based strong embedding, we can generate a 1-string representation of $G_{\mathcal{F}}$ by first generating the reduction of the single-intersection path-based strong embedding. Thus there is at most one common vertex between any two clusters due to the single-intersection-property. We can then take an arbitrary embedding of a planar support of the reduction that induces the single-intersection path-based strong embedding and generate the strings by combining the curves of the edges of the subgraph that is induced by a cluster. □

Now we can introduce two special kinds of single-intersection path-based strong embeddings. They require that the orderings of clusters on a path in a support are consistent throughout all clusters. The upcoming two classes are only defined for families of two clusterings.

4.2.6. Cylinder- and Plane-Grid Strong Embeddability

We now define strong embeddability for the case that the cluster-graph has a grid representation both on the cylinder and in the plane:

Definition 27 (Cylinder-Grid Strong Embeddings) *A family of two strict partitioning clusterings $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ on a set V is* cylinder-grid strongly embeddable *if its cluster-graph has a cylindrical grid representation. An embedding where clusters are drawn as horizontal and vertical ribbons on a cylinder is called* cylinder-grid strong embedding *(CGSE).*

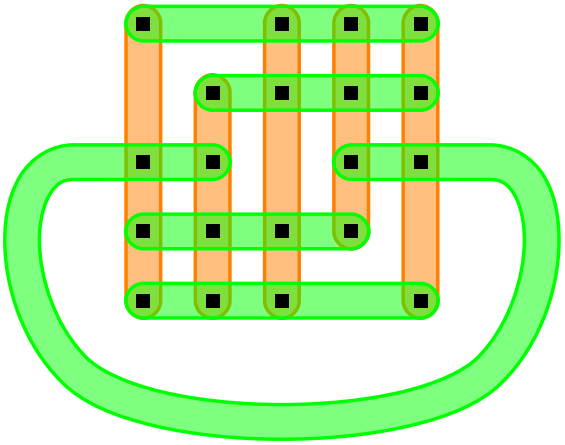


Figure 4.15.: A cylinder-grid strong embedding of a family of two clusterings

To transfer the drawing into the plane, the cylinder has to be cut between two clusters. These clusters have to be drawn as curves around the grid as shown in Figure 4.15.

It is still an open question whether we can visualize every family of clusterings that is path-based single-intersection strongly embeddable in this way. This is the only remaining unknown inclusion in our – otherwise strict – hierarchy of embeddability (Section 4.4).

If we want to avoid the cuts in the cylinder in order to draw the clusters in the plane, we can also define an even stricter concept of strong embeddability, the plane-grid strong embeddability:

Definition 28 (Plane-Grid Strong Embeddings) *A family of two strict partitioning clusterings $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ on a set V is* plane-grid strongly embeddable *if its cluster-graph has a grid representation in the plane. An embedding where clusters are drawn as horizontal and vertical ribbons in the plane is called* plane-grid strong embedding *(PGSE).*

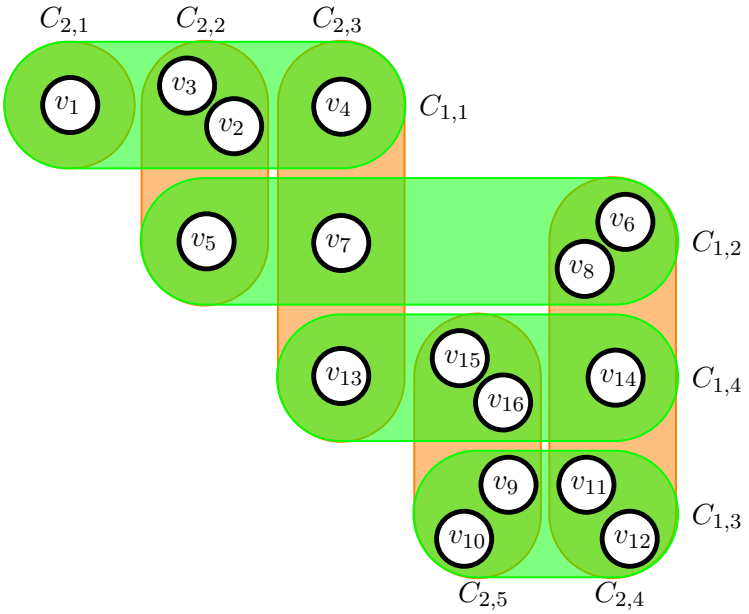


Figure 4.16.: A plane-grid strong embedding of two families of clusterings. The clusterings are the same as in the example from Fig. 3.1

Figure 4.16 shows the example we used in Chapter 3, drawn in a *plane-grid strong embedding*. Not every graph that is cylinder-grid strongly embeddable is also plane-grid strongly embeddable. The clusterings in Fig. 4.15 are such an example: The five missing vertices in the intersection of the clusters make it impossible to draw the clusters in a plane-grid without violating the requirements for strong embeddability. Since there are only four sides of a plane-grid, only four of the missing vertices can be placed at the border of the grid. The fifth missing vertex has to be placed somewhere in the middle of the grid in all possible permutations of the rows and columns, requiring at least one ribbon to be wrapped around the cylinder.

A nice advantage of plane-grid embeddings is that they lead directly to a representation where each cluster can be represented as a convex area. This should further increase the readability of embeddings.

As stated in Section 2.3.3, it is $\mathcal{NP}$-complete to decide whether a bipartite graph has a grid representation, both in the plane and on a cylinder. Thus, since every bipartite graph can be seen as a cluster-graph, the hardness also applies for the test whether two clusterings have a plane- or cylinder-grid embedding.

We will now generalize the problem from a decision- to an optimization-problem. In order to do that, we define grid representations of clusterings that are allowed to have crossings between disjoint clusters.

Definition 29 (Grid Representations of Clusterings) *A* grid representation[1] *of two clusterings in the plane or on the cylinder is a representation of the clusters as horizontal and vertical ribbons where ribbons are allowed to cross even if their corresponding clusters do not share a common vertex.*

A grid representation can be described entirely by a simple permutation of the clusters of each clustering. The search for good grid representations naturally leads to the following definition:

Definition 30 (Bad Crossings and Optimal Grid Representations) *A* bad crossing *in a cylinder- or plane-grid representation of two clusterings is an intersection between the ribbons of two clusters that do not contain a common vertex. A cylinder- or plane-grid representation of two clusterings is* optimal *if the number of bad crossings is minimal.*

Thus, every plane- or cylinder-grid embedding is an optimal plane- or cylinder-grid representation (since the number of bad crossings can not be less than zero). More generally, we can now use two numerical values for families of two clusterings that indicate how well they can be embedded.

Definition 31 (MinPlaneCrossingNumber and MinCylinderCrossingNumber) *Given two clusterings, the* MinPlaneCrossingNumber *is the number of bad crossings in an optimal plane-grid representation and the* MinCylinderCrossingNumber *the number of bad crossings in an optimal cylinder-grid representation of the two clusterings.*

In the rest of this work, we are going to put an emphasis on grid representations. They have the advantage that they are easy to describe and to represent. Before we take a closer look at grid representations, we are going to conclude this chapter by establishing the final and strictest concept of embeddability and then recapitulate the entire hierarchy of embeddability.

4.3. Full Embeddability

When looking at the concepts we established so far the question arises whether it is possible to completely avoid crossings between the insides of clusters, no matter if they have common vertices or not. We establish this as a requirement for the concept of full embeddability. This assures that a cluster is drawn in one piece.

Definition 32 (Full Embeddings) *The embedding* Γ *is a* full embedding *(FE) if it is a strong embedding and every two curves share at most two points of intersection:*

$$\forall C_a, C_b \in \bigcup_{\mathcal{C}\in\mathcal{F}} \mathcal{C} : |J(C_a) \cap J(C_b)| \leq 2$$

We call $\mathcal{F}$ fully embeddable *if there exists a full embedding for* $\mathcal{F}$*.*

For the case of two clusterings, we can establish the following relationship to Zykov planarity.

Theorem 9 *A family of two clusterings* $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ *on a set* V *is fully embeddable if and only if the condensation* $H'_{\mathcal{F}} = (V,' \mathcal{S})$ *of the corresponding hypergraph* $H_{\mathcal{F}} = (V, \mathcal{S})$ *is Zykov planar.*

[1] Please note the different context in which we are now using the term grid representation: A grid representation of two clusterings is allowed to have crossings, a grid representation of a bipartite graph not. The latter corresponds to a grid embedding when viewed as cluster-graph.

Proof

Let Γ be a full embedding of $\mathcal{F}$. Every region of the embedding lies either in the inside of two, one or zero Jordan curves. We place a vertex inside the regions that lie in the insides of two or one curve and connect two vertices if their regions are separated by only one Jordan curve. The result is a planar bipartite map, and thus the hypergraph is Zykov planar.

Let $H_{\mathcal{F}} = (V, \mathcal{S})$ be Zykov planar. There exists thus a planar embedding of its bipartite map. We generate a full embedding by drawing an enclosing curve around every star that consists of the node for a cluster and its neighbors. If we choose the distance of the enclosing curve small enough, we end up with a full embedding. □

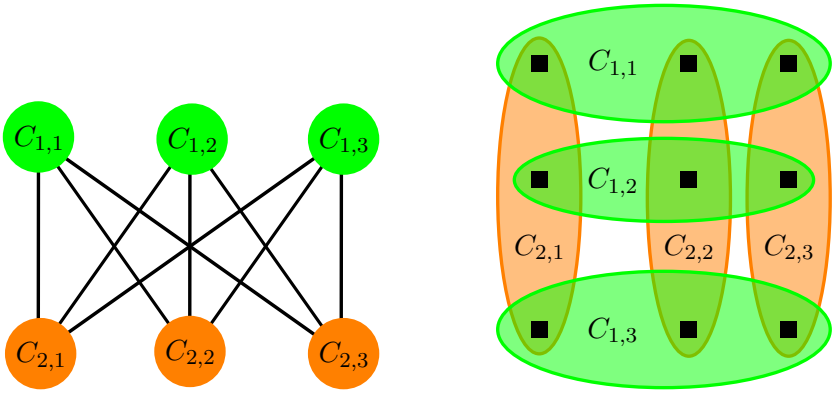
For the case of two clusterings, we show the following relationship to the planarity of the cluster-graph.

Theorem 10 *A family of two strict partitioning clusterings $\mathcal{F} = \{\mathcal{C}_1, \mathcal{C}_2\}$ is fully embeddable if and only if the cluster-graph $G_{\mathcal{F}}$ is planar.*

Proof

We generate a graph G_b by dividing every edge of $G_{\mathcal{F}}$ and inserting a node v. Since $G_{\mathcal{F}}$ is planar, G_b is also planar and – due to its construction – a bipartite map of the corresponding condensed hypergraph $H'_{\mathcal{F}}$. As described in Section 2.2.2, the existence of such a bipartite map induces the Zykov-planarity of $H'_{\mathcal{F}}$. According to Theorem 9, this is equivalent to the existence of a full embedding of $\mathcal{F}$.

For the other direction, a full embedding of $\mathcal{F}$ induces the existence of a bipartite map G_b for the condensed hypergraph $H'_{\mathcal{F}}$. From G_b we construct the cluster-graph by replacing all paths between two nodes that represent clusters with an edge. □



(a) $K_{3,3}$ as cluster-graph of $\mathcal{F}$ (b) A strong embedding of $\mathcal{F}$

Figure 4.17.: The complete bipartite graph $K_{3,3}$ and a strong embedding of two clusterings $\mathcal{F} = \{C_1, C_2\}$ induced by $K_{3,3}$ as cluster-graph. A full embedding of $\mathcal{F}$ can not exist.

Theorem 10 allows us to characterize the smallest non-fully-embeddable clusterings. The smallest non-planar Graphs are K_5 and $K_{3,3}$. Since K_5 is not k-partite, it can not be directly used as a cluster-graph. (A version of K_5 with subdivided edges as clustergraph is shown earlier in Fig. 4.3.) $K_{3,3}$ however is already bipartite and can be used as a

cluster-graph. Figure 4.17 shows $K_{3,3}$ as a cluster-graph and a strong embedding of the clusterings. A full embedding can not exist.

Furthermore, we can show that the instances of families of two clusterings that are fully embeddable are a true subset of the instances that are plane-grid strongly embeddable: As mentioned in Section 2.3.3, Hartman et al. [HNZ91] showed that every planar bipartite graph has a grid representation in the plane. Since the bipartite cluster-graph of every fully embeddable family of two clusterings is planar, it has a grid representation. This is exactly the definition of plane-grid strong embeddability (Definition 28).

Now we can generate an overview of the relationships between the classes we introduced.

4.4. The Hierarchy of Embeddability

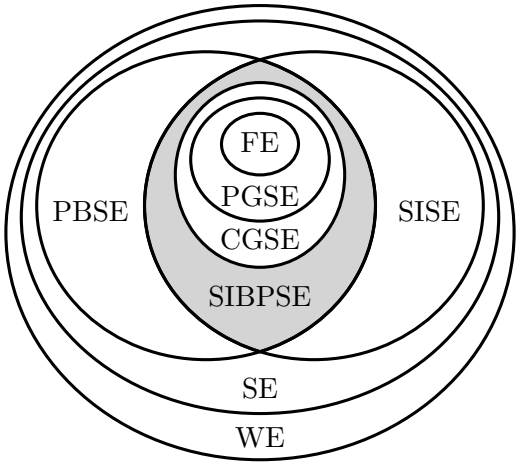


Figure 4.18.: The hierarchy of embeddability for families of two clusterings

After having introduced all classes of embeddability, we will summarize our results in a hierarchy of embeddability. Figure 4.18 visualizes the relationship between the different classes for the case of two clusterings. We could show that all classes are true subsets of each other, with one exception: It is not clear whether there exist embeddings that have a single-intersection path-based strong embedding but are not cylinder-grid strongly embeddable. In other words, it is not clear whether single-intersection path-based strong embeddability is its own class or not. We therefore shaded this class gray in the visualization.

Figure 4.19 shows again the differences between the main classes of weak, strong, and full embeddings.

For a more detailed overview of all classes, we provide Table 4.2. It summarizes all important results of this work for the case of two clusterings, regarding the relationship to other problems and the complexity of the decision problem for each class.

We give references to the corresponding lemmata and theorems if we proved a result in this work or cite other publications if we use their results. For the inclusions, we refer to figures that help to visualize the differences between the two classes.

Full Embeddability
$\Leftrightarrow$ condensed hypergraph is Zykov planar (Theorem 9) $\Leftrightarrow$ cluster-graph is planar (Theorem 10)
linear complexity (planarity test)

$\cap$ Fig. 4.17

Plane-Grid Strong Embeddability
$\Leftrightarrow$ clustergraph is grid intersection graph in the plane (Def. 28)
$\mathcal{NP}$-complete (see [Kra94])

$\cap$ Fig. 4.15

Cylinder-Grid Strong Embeddability
$\Leftrightarrow$ clustergraph is grid intersection graph on a cylinder (Def. 27)
$\mathcal{NP}$-complete (see [KP96])

$\cap$? $\parallel$

Single-Intersection Path-Based Strong Embeddability
$\Leftrightarrow$ both single-intersection and path-based strongly embeddable (Definition 26) $\Leftrightarrow$ cluster-graph is in 1-string (Corollary 7)
complexity unknown

$\cap$ $\cap$

Single-Intersection Strong Embeddability	**Path-Based Strong Embeddability**
$\Leftrightarrow$ condensation of corresponding hypergraph is strongly embeddable (Theorem 7)	$\Leftrightarrow$ clustergraph is individually bounded intersection string-graph (Theorem 8)
complexity unknown	complexity unknown

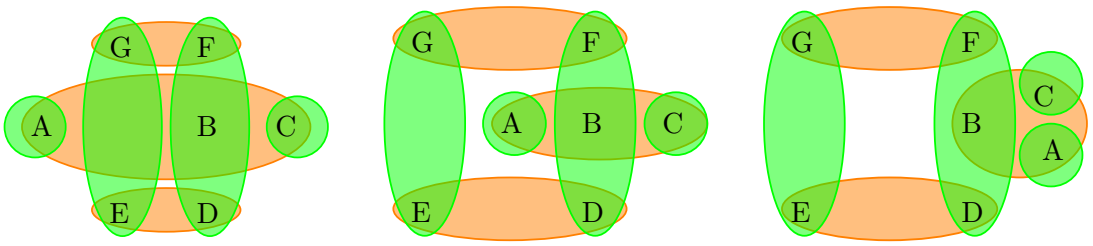
$\cap$ Fig. 4.13(a) $\cap$ Fig. 4.13(c)

Strong Embeddability
$\Rightarrow$ clustergraph is a string graph (Lemma 1) $\Leftrightarrow$ corresponding hypergraph is vertex planar (Theorem 4)
$\mathcal{NP}$-complete (Theorem 6)

$\cap$ Fig. 4.3

Weak Embeddability
$\Leftrightarrow$ the clusterings have to be strict partitioning (Theorem 3)
exists always

Table 4.2.: An overview of the results from this chapter for the case of two clusterings



(a) Weak embedding (b) Strong embedding (c) Full embedding

Figure 4.19.: A comparison of the three main types of embeddings on the example clusterings from Fig. 1.1

5. How to Generate Embeddings

In this chapter we are going to introduce practical approaches for generating embeddings for families of two clusterings. The main goal is to generate embeddings with as few intersections as possible between the insides of curves of clusters that do not share a common vertex.

Since this work focuses on embeddability, we did not put an emphasis on visual appearance or aesthetic criteria, such as convex shapes of clusters or compactness of the representation. Instead, we simply generate a layout of a support of the clusterings and draw enclosing curves around the subgraphs induced by the clusterings.

Thus, the methods presented in this chapter have the goal of finding a support. The optimum would be to to find a support that is planar, leading to a strong embedding of the clusterings. If we can not find a planar support, we will try to provide a support that can be embedded with as few crossings as possible. This is what we refer to as good support. In order to layout the supports, we used layout algorithms that are included in the "Open Graph Drawing Framework" (OGDF, see [CGJ+13]). We additionally implemented a visualization for plane- and cylinder-grid representations. Information on the format of the input for our implementation can be found in Section A in the Appendix.

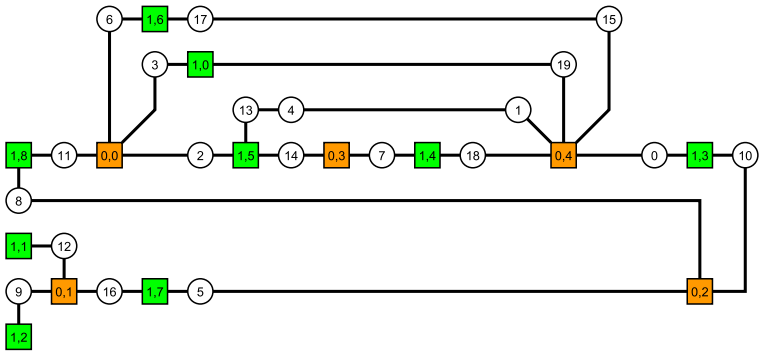
For fully embeddable families of clusterings, finding a planar support is easy (Section 5.1). For a non-fully embeddable family of clusterings however, it is $\mathcal{NP}$-complete to decide whether there exists a planar support or not (see proof in Section 4.2.4). Yet we are not only interested in the decision problem but also in the related optimization problem, i.e., minimizing crossings in an embedding of a support. Obviously this problem is even harder since the decision problem is just a special case of the optimization problem.

We were not able to provide a simple method to directly approach this problem besides random guessing. Yet we developed both a heuristic and an exact method to minimize crossings in cylinder- and plane-grid representations. With these results, we can either output the grid representation itself or generate a support that is based on this representation.

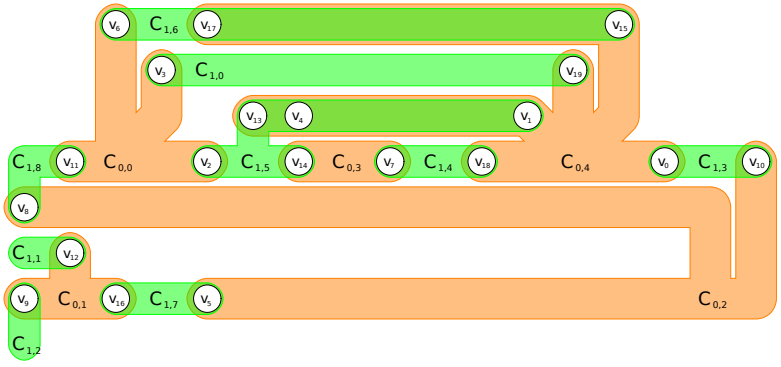
For technical reasons, we add a dummy-node for each cluster to the support that serves as a placeholder for the label of the cluster. We start with the easy case of generating a planar support for families of clusterings that are fully embeddable.

5.1. Supports for Fully Embeddable Families of Clusterings

Fully embeddable families of clusterings induce a planar graph G_b, the bipartite map (Section 2.2). We can create a support by connecting equivalent nodes to a chain and then create an edge from one of the nodes in the equivalence class to the dummy node of the containing cluster. Figure 5.1 shows a layout of such a support and the corresponding output of our program for an example instance with 20 vertices[1].



(a) A layout of the generated planar support based on a bipartite map



(b) The resulting full embedding of the family of clusterings

Figure 5.1.: The support and the resulting full embedding of a fully embeddable family of clusterings generated by our implementation

5.2. Supports for Non-Fully Embeddable Families of Clusterings

If the cluster-graph is not planar, the search for a good support becomes more complicated. If a cluster contains n vertices, there exist n^{n-2} different spanning trees that cover the vertices of the cluster (see Cayley's formula [Cay89]). Even if we restrict the search space to path-based supports, we still have $n!$ possible permutations of the n vertices on the path that forms the support for a cluster.

Due to the $\mathcal{NP}$-completeness result in Section 4.2.4, it is unlikely that there exists a polynomial-time algorithm for finding a support. For the case of a path-based support, we did not prove the hardness; yet there is at least no obvious way to tackle the problem. A brute-force approach that tries all possible combinations is not practicable due to the

[1] Note that – contrary to the examples in earlier chapters – our method numbers clusterings and clusters starting with the index 0 instead of 1

exponential number of possibilities. However, for experiments with smaller examples, we implemented two random-support generators:

Random Path-Based Support

We generate a random order of all vertices in a cluster and connect them to a path-based support. We then check if the support is planar and restart with another random order if it is not.

Random Minimal Support

We use the same principle as for the random path support, but we are generating a random spanning tree for the vertices in each cluster instead of a random path. We use a simple algorithm that was proposed by Wilson [Wil96] to generate spanning trees uniformly at random.

The random methods provide good results for very small instances, especially if the solution space is large (i.e., there exist many planar supports) and the clusters are rather small. Figure 5.2 shows embeddings that are induced by a random path support and a random minimal support for the same example we used earlier.

While it is already not straight-forward to find a heuristic that finds a planar support, the problem becomes even harder when it comes to generating a good support (i.e., minimize the crossings). Already the comparison between two given non-planar supports is a hard problem: Garey and Johnson [GJ83] proved the problem of finding the minimal crossing number of a graph to be $\mathcal{NP}$-complete. That means that even if we have two non-planar supports, it is hard to tell which one is better, i.e., which one can be embedded with the least number of crossings.

We therefore decided to focus on supports that are induced by cylinder-grid representations. These are easier to describe, and the solution space is covered by all possible permutations of the rows and columns in the grid.

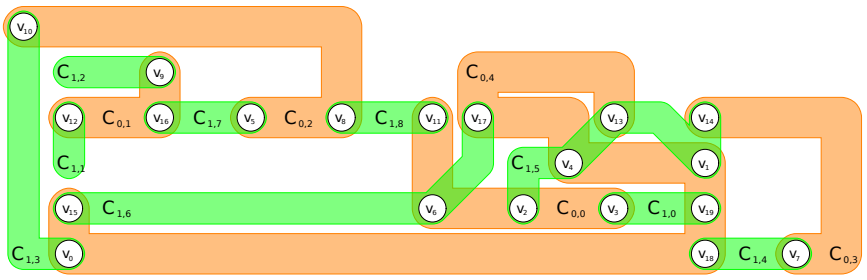
Cylinder-Grid Induced Support

From a cylinder-grid representation of two clusterings we can generate a support by connecting equivalent nodes to a chain and connect the chains in the order of their appearance in the ribbon that represents the cluster. For bad crossings we can introduce a dummy-node, such that the result is a planar graph for which we can again draw our enclosing curves. This assures that the support has at most k crossings if there exist k crossings in the cylinder-grid representation. Figure 5.3 shows a plane- and a cylinder-grid representation of the example-clusterings as well as an embedding based on the support that is induced by the cylinder-grid representation.

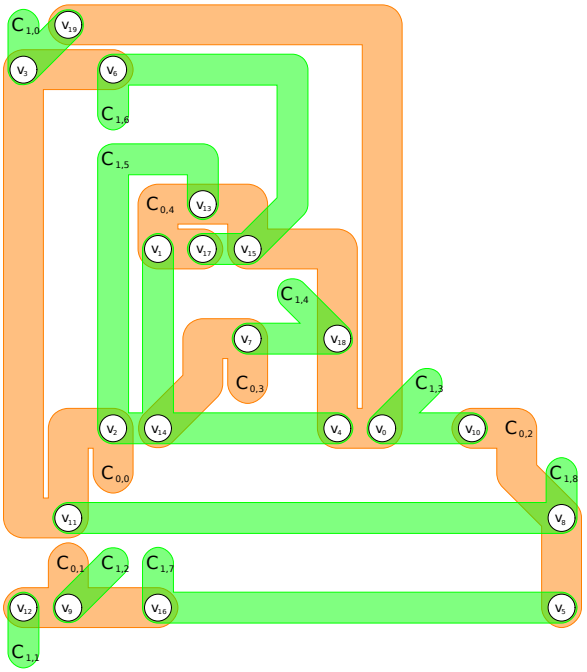
Now the main question is how to find a good or even optimal cylinder-grid representation. In the following section we will introduce two different approaches for generating grid representations.

5.3. How to Find Grid Representations

We developed two methods for finding grid representations of two clusterings. As we showed earlier, the corresponding decision problem is $\mathcal{NP}$-complete. The first approach is a greedy heuristic that is fast but is not guaranteed to provide an optimal result. The second approach generates an optimal representation by modeling the problem as an integer linear program (ILP).



(a) A strong embedding based on a random path support



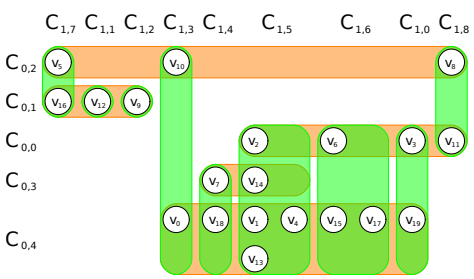
(b) A strong embedding based on a random minimal support

Figure 5.2.: Two strong embeddings based on randomly generated supports

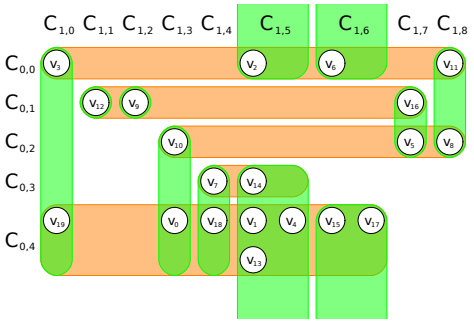
A linear program is an optimization problem where a linear objective function has to be maximized or minimized under a series of constraints that have to be linear as well (See [FS58] for an overview). While there exist efficient solutions for the general case, i.e., all variables can have real values, the problem becomes $\mathcal{NP}$-complete if we limit the solution space to integer or binary values (see [Pap81]). We are thus transforming one $\mathcal{NP}$-complete problem into another. The advantage is that there exist fast solvers for ILP's that use a combination of heuristics and exhaustive search in the solution space to solve the problem. However, there is no guarantee to find such a solution quickly, and running times can increase exponentially with the size of the problem.

5.3.1. A Greedy Heuristic to Find a Grid Representation

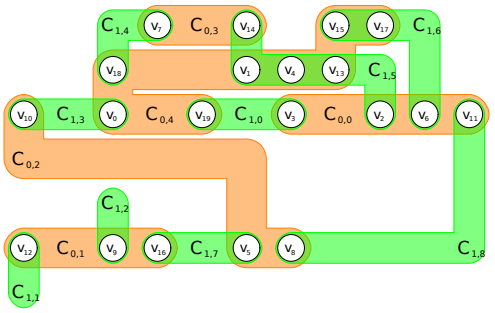
In our heuristic we start with a grid representation induced by an arbitrary permutation of the rows and columns. We then alternatingly swap two random rows or columns until we reach the maximal number of iterations or find a representation without bad crossings. If the swapping increased the number of bad crossings, we undo the swap, otherwise we keep the new grid representation. This method works with both cylinder- and plane-grid



(a) An optimal plane-grid representation



(b) An optimal cylinder-grid representation



(c) A strong embedding based on the support that was induced by the cylinder-grid representation above

Figure 5.3.: Two grid representations and a strong embedding based on a cylinder-grid induced support

representations, depending on the function *currentCrossingNum()* used to calculate the number of crossings. The pseudocode in Algorithm 1 describes the heuristic.

Algorithm 1: Greedy Heuristic for finding a grid representation

Input : two clusterings $\mathcal{C}_0, \mathcal{C}_1$; number of maximal iterations m
Output: a grid representation
generate initial grid representation
minCrossings ← currentCrossingNum()
while $m > 0$ *AND* *minCrossings* > 0 **do**
  if *isEven(m)* **then** swapTwoRandomRows()
  else swapTwoRandomCols()
  if *minCrossings* $<$ *currentCrossingNum()* **then** undoSwapping()
  else minCrossings ← currentCrossingNum()
  $m \leftarrow m - 1$

The methods for calculating the number of bad crossings for plane- and cylinder-grid representations can be found in Section B of the Appendix.

In the following two sections we introduce two integer linear programs for finding optimal plane- and cylinder-grid representations. In Section 6.1 we will compare results of the greedy approach with exact solutions from the integer linear programs.

5.3.2. Integer Linear Programs for Optimal Grid Representations

The following two integer linear programs (ILP) allow to calculate an optimal solution of the MinCylinderCrossingNumber- and the MinPlaneCrossingNumber-problem. We begin with the easier case of plane-grid representations.

5.3.2.1. Optimal Plane-Grid Representations

Let $\mathcal{C}_A$ and $\mathcal{C}_B$ be the two clusterings and $n = |\mathcal{C}_A|$ and $m = |\mathcal{C}_B|$ be their sizes. We will assume that we draw the clusters in $\mathcal{C}_A$ vertically (as columns) and the clusters in $\mathcal{C}_B$ horizontally (as rows). We use the prefixes `A` and `B` to indicate whether an index in a variable corresponds to the first or the second clustering. We call the crossings between the clusters *grid cells*. A grid cell is *empty* if the two clusters that cross in the cell do not share a common vertex.

The Variables

We generate binary variables `A`i`InPos`x (for $0 \leq i < n$ and $0 \leq x < n$) and binary variables `B`j`InPos`y (for $0 \leq j < m$ and $0 \leq y < m$) that indicate – if set to 1 – that the cluster i of $\mathcal{C}_A$ is in position x of the rows and that the cluster j of $\mathcal{C}_B$ is in position y of the columns.

We further generate integer variables `A`i`Pos` (for $0 \leq i < n$) and `B`j`Pos` (for $0 \leq j < m$) that indicate the row and column indices of the clusters for both clusterings.

For every cluster, we generate boundaries `A`i`FirstNonZero` and `A`i`LastNonZero` (for $0 \leq i < n$) for the clusters in the first clustering and `B`j`FirstNonZero` and `B`j`LastNonZero` (for $0 \leq j < m$) for the clusters in the second clustering. The boundaries indicate the position of the first and the last grid cell that is not empty in the row or column of the cluster.

For every pair of clusters $C_i \in \mathcal{C}_A$ and $C_j \in \mathcal{C}_B$ that does not contain a common vertex, we also generate the following binary *boundary-indicator-variables*:

`A`i`B`j`GreaterThanFirstNonZeroA`
`A`i`B`j`SmallerThanLastNonZeroA`
`A`i`B`j`GreaterThanFirstNonZeroB`
`A`i`B`j`SmallerThanLastNonZeroB`
(for $0 \leq i < n$ and $0 \leq j < m$ if C_i does not share a vertex with C_j)

They indicate whether the position of the empty grid cell is greater or smaller than the boundaries in the corresponding columns or rows.

We also create a binary variable `A`i`B`j`HasBadCrossing` (for $0 \leq i < n$ and $0 \leq j < m$) for every empty grid cell. This variable will be set to true if the empty cell lies within the boundaries of both containing clusters, inducing a bad crossing.

The Constraints

In order to guarantee that every position is only taken by one cluster, we introduce the following conditions for the two clusterings:

$$\sum_{i=0}^{n} \texttt{A}i\texttt{InPos}x = 1 \text{ (for } 0 \leq x < n)$$

$$\sum_{j=0}^{m} \texttt{B}j\texttt{InPos}y = 1 \text{ (for } 0 \leq y < m)$$

We also have to make sure that every cluster is in exactly one position:

$$\sum_{x=0}^{n} \texttt{A}i\texttt{InPos}x = 1 \text{ (for } 0 \leq i < n)$$

$$\sum_{y=0}^{m} \texttt{B}j\texttt{InPos}y = 1 \text{ (for } 0 \leq j < m)$$

Since only one of the binary variables can be 1, we can calculate the integer-positions as follows:

$$\texttt{A}i\texttt{Pos} = \sum_{x=0}^{n} x \cdot \texttt{A}i\texttt{InPos}x \text{ (for } 0 \leq i < n)$$

$$\texttt{B}j\texttt{Pos} = \sum_{y=0}^{m} y \cdot \texttt{B}j\texttt{InPos}y \text{ (for } 0 \leq j < m)$$

We now make sure that the boundaries are valid by establishing the following conditions for every pair of clusters that contains a common vertex:

$$\begin{aligned} \texttt{A}i\texttt{FirstNonZero} &\leq \texttt{B}j\texttt{Pos} \\ \texttt{A}i\texttt{LastNonZero} &\geq \texttt{B}j\texttt{Pos} \\ \texttt{B}j\texttt{FirstNonZero} &\leq \texttt{A}i\texttt{Pos} \\ \texttt{B}j\texttt{LastNonZero} &\geq \texttt{A}i\texttt{Pos} \\ \text{(for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ shares a vertex with } C_j) & \end{aligned}$$

For the remaining pairs of clusters that do not contain a common vertex, we compare their positions to the boundaries and set the binary boundary-indicator-variable to true (1) if the condition is fulfilled. The "Big-M"-method (see [GNS09]) forces the indicator-variable to be equal to 1, in the case that the condition is greater than zero. Otherwise, the indicator-variable can be 0. From now on we choose an M that is large enough, for example $M := max(n, m) + 1$.

$$\begin{aligned} \texttt{B}j\texttt{Pos} - \texttt{A}i\texttt{FirstNonZero} - M \cdot \texttt{A}i\texttt{B}j\texttt{GreaterThanFirstNonZeroA} &\leq -1 \\ \texttt{A}i\texttt{LastNonZero} - \texttt{B}j\texttt{Pos} - M \cdot \texttt{A}i\texttt{B}j\texttt{SmallerThanLastNonZeroA} &\leq -1 \\ \texttt{A}i\texttt{Pos} - \texttt{B}j\texttt{FirstNonZero} - M \cdot \texttt{A}i\texttt{B}j\texttt{GreaterThanFirstNonZeroB} &\leq -1 \\ \texttt{B}j\texttt{LastNonZero} - \texttt{A}i\texttt{Pos} - M \cdot \texttt{A}i\texttt{B}j\texttt{SmallerThanLastNonZeroB} &\leq -1 \\ \text{(for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ does not share a vertex with } C_j) & \end{aligned}$$

Now we can calculate the indicator-variable that shows whether there exists a bad crossing in an empty grid cell. If all four boundary-indicator-variables are 1, the empty grid cell lies in the crossing of two ribbons, and the `HasBadCrossing`-variable has to be 1 in order to fulfill the equation. If one or more of the boundary-indicator-variables are 0, the cell lies outside of at least one of the ribbons and can be set to 0.

$$\begin{aligned} \texttt{A}i\texttt{B}j\texttt{GreaterThanFirstNonZeroA} + \texttt{A}i\texttt{B}j\texttt{SmallerThanLastNonZeroA} & \\ +\texttt{A}i\texttt{B}j\texttt{GreaterThanFirstNonZeroB} + \texttt{A}i\texttt{B}j\texttt{SmallerThanLastNonZeroB} & \\ -\texttt{A}i\texttt{B}j\texttt{HasBadCrossing} \leq 3 & \\ \text{(for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ does not share a vertex with } C_j) & \end{aligned}$$

The Objective Function

The objective is simple: minimize the number of bad crossings:

$$\text{minimize} \sum_{C_i \cap C_j = \emptyset} \texttt{A}i\texttt{B}j\texttt{HasBadCrossing}$$

$$(\text{for } C_i \in \mathcal{C}_A \text{ and } C_j \in \mathcal{C}_B,\ 0 \leq i < n \text{ and } 0 \leq j < m)$$

Proposition 4 *There exists a solution of value k for the ILP if and only if there exists a plane-grid representation of the two clusterings with k bad crossings.*

Proof

Let us assume that there exists a solution of the ILP with value k. When transferring the permutations of the rows and columns to a plane-grid representation, we can draw the ribbons according to the boundaries induced by the solution of the ILP. The calculation of the indices of the boundaries guarantees that every non-empty crossing lies between the boundaries of its corresponding row or column and that we thus end up with a valid representation. Due to the construction of the ILP, every bad crossing in the grid representation augments the solution by 1. Thus, there exists a plane-grid representation with at most k bad crossings.

Assuming we have a grid representation with k crossings. We can transfer the indices of the rows and columns as well as the boundaries of the ribbons to the ILP. Due to the construction of the ILP, we can set all `HasBadCrossing`-variables for empty grid cells that do not induce a bad crossing to 0. Thus the solution of the ILP corresponds to the number of bad crossings. □

5.3.2.2. Optimal Cylinder-Grid Representations

For the case of a cylinder-grid representation, the program becomes a bit more complex since we have to take into account that rows or columns wrap behind the cylinder. We modeled this wrapping by setting the upper limit of a ribbon smaller than its lower limit. In the following we show how the linear program above can be modified and extended to allow wrapped rows or columns.

The Variables

Additionally to the variables used for the linear program in the section above, we introduce the following new variables:

For every pair of clusters $C_i \in \mathcal{C}_A$ and $C_j \in \mathcal{C}_B$ that contain a common vertex, we add four boundary-indicator-variables variables that – contrary to the ones we added for the clusters without a common vertex – indicate if the corresponding grid cell lies outside of the boundaries:

$$\begin{aligned} &\texttt{A}i\texttt{B}j\texttt{SmallerThanFirstNonZeroA} \\ &\texttt{A}i\texttt{B}j\texttt{GreaterThanLastNonZeroA} \\ &\texttt{A}i\texttt{B}j\texttt{SmallerThanFirstNonZeroB} \\ &\texttt{A}i\texttt{B}j\texttt{GreaterThanLastNonZeroB} \\ &(\text{for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ shares a vertex with } C_j) \end{aligned}$$

For every pair of clusters that does not contain a common vertex, we additionally generate the following binary variables to indicate whether they are within the boundaries of the

ribbon in a row or column, respectively: AiBjWithinBoundsA and AiBjWithinBoundsB (for $0 \leq i < n$ and $0 \leq j < m$ if C_i does not share a vertex with C_j)

We also declare the binary variables AiIsWrapped (for $0 \leq i < n$) and BjIsWrapped (for $0 \leq j < m$) that indicate if a row or column is wrapped, i.e., runs behind the cylinder and reenters the grid from the other side.

Finally, we introduce the two binary variables RowsAreWrapped and ColsAreWrapped that indicate if there exist one or more rows or cols that are wrapped.

The Constraints

We use the same inequalities as above to describe the positions of the clusters and to generate the boundary-indicator-variables for the empty grid cells.

However, we do not generate the boundaries for the clusters directly anymore: We now use the boundary-indicator-variables for the non-empty grid cells:

$$\begin{aligned} \texttt{A}i\texttt{B}j\texttt{SmallerThanFirstNonZeroA} + \texttt{A}i\texttt{B}j\texttt{GreaterThanLastNonZeroA} - \texttt{A}i\texttt{IsWrapped} &= 0 \\ \texttt{A}i\texttt{B}j\texttt{SmallerThanFirstNonZeroB} + \texttt{A}i\texttt{B}j\texttt{GreaterThanLastNonZeroB} - \texttt{B}j\texttt{IsWrapped} &= 0 \\ \text{(for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ shares a vertex with } C_j\text{)} & \end{aligned}$$

If a row or column is not wrapped, every non-empty cell lies inside of both boundaries of the cluster. Thus, both boundary-indicator-variables are equal to 0 (due to the inverted indicators for non-empty grid cells). If a row is wrapped, it lies inside of exactly one of the boundaries, and the other indicator-variable is 1, fulfilling the equation.

To generate the actual boundary-variables, we introduce the following constraints:

$$\begin{aligned} \texttt{A}i\texttt{FirstNonZero} - \texttt{B}j\texttt{Pos} - M \cdot \texttt{A}i\texttt{B}j\texttt{SmallerThanFirstNonZeroA} &\leq 0 \\ \texttt{B}j\texttt{Pos} - \texttt{A}i\texttt{LastNonZero} - M \cdot \texttt{A}i\texttt{B}j\texttt{GreaterThanLastNonZeroA} &\leq 0 \\ \texttt{B}j\texttt{FirstNonZero} - \texttt{A}i\texttt{Pos} - M \cdot \texttt{A}i\texttt{B}j\texttt{SmallerThanFirstNonZeroB} &\leq 0 \\ \texttt{A}i\texttt{Pos} - \texttt{B}j\texttt{LastNonZero} - M \cdot \texttt{A}i\texttt{B}j\texttt{GreaterThanLastNonZeroB} &\leq 0 \\ \text{(for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ shares a vertex with } C_j\text{)} & \end{aligned}$$

They guarantee that the boundary-indicator-variables of the non-empty cells are set to 1 if they lie outside of the corresponding boundary and therewith implicitly set the boundary.

We now have to set the indicator-variable whether a row or a column is wrapped. The following construction guarantees that the indicator-variable is set to 1 if and only if the upper boundary is smaller than the lower boundary.

$$\begin{aligned} \texttt{A}i\texttt{FirstNonZero} - \texttt{A}i\texttt{LastNonZero} - M \cdot \texttt{A}i\texttt{IsWrapped} &\leq 0 \text{ (for } 0 \leq i < n\text{)} \\ \texttt{A}i\texttt{FirstNonZero} - \texttt{A}i\texttt{LastNonZero} - M \cdot \texttt{A}i\texttt{IsWrapped} &\geq 1 - M \text{ (for } 0 \leq i < n\text{)} \\ \texttt{B}j\texttt{FirstNonZero} - \texttt{B}j\texttt{LastNonZero} - M \cdot \texttt{B}j\texttt{IsWrapped} &\leq 0 \text{ (for } 0 \leq j < m\text{)} \\ \texttt{B}j\texttt{FirstNonZero} - \texttt{B}j\texttt{LastNonZero} - M \cdot \texttt{B}j\texttt{IsWrapped} &\geq 1 - M \text{ (for } 0 \leq j < m\text{)} \end{aligned}$$

We check whether an empty cell lies within the boundaries as follows:

$$\begin{aligned} \texttt{A}i\texttt{B}j\texttt{GreaterThanFirstNonZeroA} + \texttt{A}i\texttt{B}j\texttt{SmallerThanLastNonZeroA} + \\ \texttt{A}i\texttt{IsWrapped} - \texttt{A}i\texttt{B}j\texttt{WithinBoundsA} \leq 1 \\ \texttt{A}i\texttt{B}j\texttt{GreaterThanFirstNonZeroB} + \texttt{A}i\texttt{B}j\texttt{SmallerThanLastNonZeroB} + \\ \texttt{B}j\texttt{IsWrapped} - \texttt{A}i\texttt{B}j\texttt{WithinBoundsB} \leq 1 \\ \text{(for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ does not share a vertex with } C_j\text{)} \end{aligned}$$

If the row or column is wrapped, one true boundary-indicator-variable is sufficient for the cell to lie in the cluster, otherwise, both boundary-indicator-variables have to be 1. The within-bounds indicator-variable can thus only be 0 if the empty grid cell does not lie within the boundaries of the ribbon.

To check if there exists a bad crossing between the two clusters, we set the following conditions:

$$\texttt{A}i\texttt{B}j\texttt{WithinBoundsA} + \texttt{A}i\texttt{B}j\texttt{WithinBoundsB} - \texttt{A}i\texttt{B}j\texttt{HasBadCrossing} \leq 1$$
$$\text{(for } 0 \leq i < n \text{ and } 0 \leq j < m \text{ if } C_i \text{ does not share a vertex with } C_j\text{)}$$

If an empty cell lies within the boundaries of both a horizontal and a vertical cluster, it indicates a crossing of two clusters without a common vertex.

Now we just have to make sure that it is forbidden to have wrapped rows and columns at the same time:

$$\sum_{i=0}^{n} \texttt{A}i\texttt{IsWrapped} - M \cdot \texttt{RowsAreWrapped} \leq 0$$
$$\sum_{j=0}^{m} \texttt{B}j\texttt{IsWrapped} - M \cdot \texttt{ColsAreWrapped} \leq 0$$
$$\texttt{RowsAreWrapped} + \texttt{ColsAreWrapped} \leq 1$$

Note that if we do not add this condition, the linear program calculates the minimal number of crossings in a grid representation on a torus instead of a cylinder.

The Objective Function

The objective remains the same: minimize the number of bad crossings:

$$\text{minimize} \sum_{C_i \cap C_j = \emptyset} \texttt{A}i\texttt{B}j\texttt{HasBadCrossing}$$
$$\text{(for } C_i \in \mathcal{C}_A \text{ and } C_j \in \mathcal{C}_B,\ 0 \leq i < n \text{ and } 0 \leq j < m\text{)}$$

Proposition 5 *There exists a solution of value k for the ILP if and only if there exists a cylinder-grid representation of the two clusterings with k bad crossings.*

Proof

Let us assume that there exists a solution of the ILP with value k. When transferring the permutations of the rows and columns to a cylinder-grid representation, we can draw the ribbons according to the boundaries induced by the solution of the ILP. For the ribbons that are indicated as wrapped, we draw the ribbons in two parts. The implicit generation of the indices of the boundaries guarantees that every non-empty crossing lies between the boundaries of its corresponding row or column in the non-wrapped case and above or below one of the boundaries in the wrapped case. We thus end up with a valid representation. Due to the construction of the ILP, every bad crossing in the grid representation augments the solution by 1. Thus, there exists a plane-grid representation with at most k bad crossings.

Assuming we have a grid representation with k crossings. We can transfer the indices of the rows and columns as well as the boundaries of the ribbons to the ILP. Due to the construction of the ILP, we can set all `HasBadCrossing`-variables for empty grid cells that do not induce a bad crossing to 0. Thus the solution of the ILP corresponds to the number of bad crossings in the cylinder-grid representation. □

In the next chapter, we are going to compare the exact results from the linear program to the results we obtained from the heuristic.

6. Experimental Evaluation of Grid Representations

We conducted a series of experiments regarding plane-grid representations of randomly generated clusterings (Section 6.1) and also applied our methods to real-world data (Section 6.2).

For the random data, we used a total of 1900 test instances for which we generated cylinder- and plane-grid representations and examined the number of crossings needed in the representation. Regarding the size of the instances, we speak of a *grid size* of $n \times m$ if the sizes of the clusterings are n and m, respectively.

Besides the grid size, the main factor that influences how many crossings are needed in a grid representation is the *grid coverage* of non-empty grid cells. That is the relative number of pairs of clusters that share a common vertex. We thus talk about a grid coverage of x percent if x percent of the grid cells contain a vertex. The grid coverage is a measure on how similar the clusterings are. The higher the grid coverage, the higher the distribution of the vertices of a cluster among different clusters in the other clustering. The coverage corresponds to the number of edges in the cluster-graph divided by the number of edges in the complete bipartite graph $K_{n,m}$.

Our test instances were generated by adding vertices to random pairs of clusters in the two clusterings until we reach the desired grid coverage. The pseudocode for this method can be found in Algorithm 5 in Section C of the Appendix.

The test instances consisted of grids of sizes 10×10, 10×15, 15×15, 15×20, and 20×20. For each size we generated 20 instances for each grid coverage from 5 to 95 percent at intervals of 5 percentage points.

6.1. Embeddability Depending on Grid-Size and -Coverage

We first examine how the number of crossings in grid representations is influenced by the number of clusters and the grid coverage. Since plane-grid representations are a special case of cylinder-grid representations, the latter should have lower crossing numbers. We further expected an increase of crossings with the grid size, simply because there are more possibilities for crossings. Another assumption was that the number of crossings increases with the coverage up to a certain extend and then decreases again, since at a coverage of 100 percent, there exists a complete grid as trivial solution without bad crossings.

Due to very high running times of the ILP for the larger instances, it was not possible to generate exact solutions for grid sizes larger than 10×10. The following statistics are thus based on the results of the heuristic. Nevertheless, as we will see later, we can make the assumption that the distribution of the crossings in the exact solution would have a similar shape. For all experiments, we set a maximum of 100,000 iterations for the heuristic. This led to running times of seconds even on larger grids.

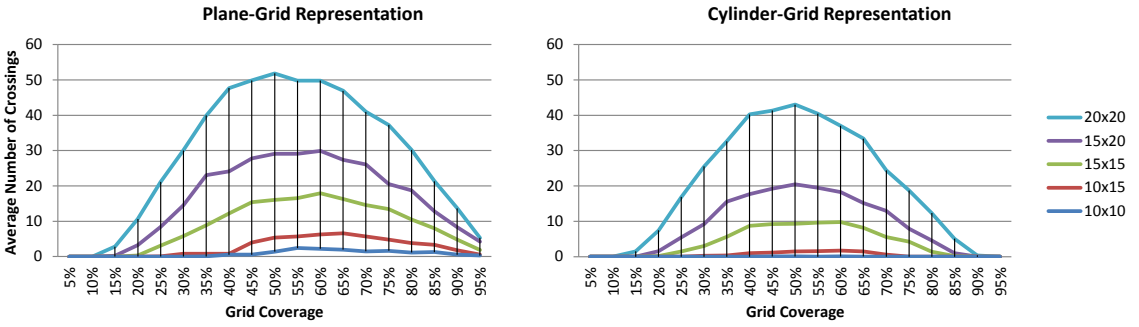


Figure 6.1.: The average number of crossings in the grid representations generated by the heuristic depending on the grid coverage

The diagrams in Fig. 6.1 show the development of the average crossing numbers of plane- and cylinder-grid representations of different sizes, depending on the grid coverage. All values are results of the heuristic and therefore not optimal. Nevertheless, we can see that the highest numbers of crossings occur around grid coverages between 40 and 65 percent. As expected, the number of crossings in the cylinder-grid representations is smaller than in the plane-grid representations. The diagrams also show that the number of crossings increases fast with the size of the grid. This reduces the likelihood of finding a planar support through a cylinder-grid representation for these instances. In the 10×10 case, the heuristic almost always found a cylinder-based embedding without crossings. In the 20×20 case, this is only true for grid coverages below 15 percent or over 90 percent. Of all instances, we measured the maximal number of 50 crossings on the cylinder-grid and 60 crossings in the plane-grid at 50 percent grid coverage in the 20×20 grid.

Now the question arises in how far the distribution of the crossing numbers generated by the heuristic correlates with the distribution of an exact solution generated by the ILP. Unfortunately we had to limit our test setting for the ILP to plane-grid representations of size 10×10. For larger grids or cylinder-grid representations, the optimization process with the Gurobi-LP-Solver [Inc13] often exceeded the time limit of 48 hours on our test-machine (2x Dual-Core Intel Xeon^{TM}E5-430 CPU (2,66 GHz) with 32 GB RAM).

In the case of the 10×10 grid however, we could calculate all optimal plane-grid representations within a reasonable timespan. Yet there was a huge variation in the times needed to solve the ILP, even between instances of the same size and grid coverage. Table D.3 in the Appendix shows the running times of Gurobi in the standard configuration on the test machine for these instances.

For the 10×10 cases that we examined, Fig. 6.2 shows the average number of crossings in the results of the heuristic compared to the optimal solution by the ILP. The raw data of this experiment containing the crossing number for every single instance can be found in Section D of the Appendix.

In more than 70 percent of these cases, the result of the heuristic corresponds to the exact solution. When taking all instances into account, the average crossing number was by 0.36

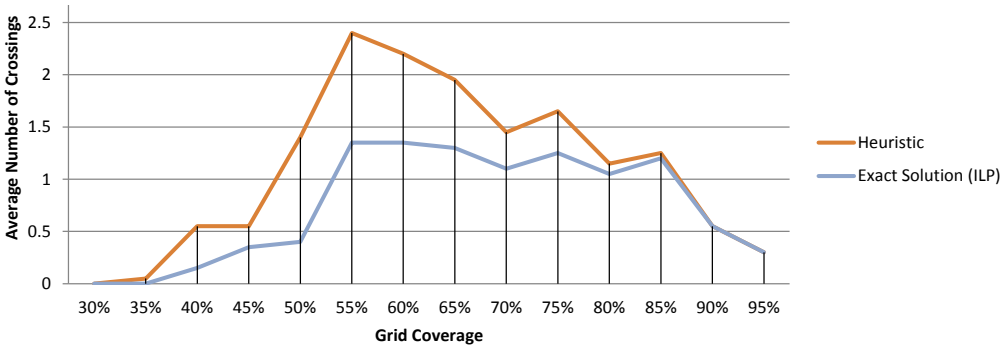


Figure 6.2.: The average crossing numbers of the representations of a 10×10 grid generated by the heuristic and the exact solution, depending on the grid coverage

higher than the exact solution. When limited to the cases where the heuristic did not provide an optimal solution, the number of crossings of the heuristic was on average 1.24 higher than the optimal solution. Since in the 10×10 case the distribution of the numbers of crossings of the heuristic seems to follow the distribution of the exact solution, it is reasonable to assume that this would be also the case for the larger instances we examined earlier.

In seven instances of the 10×10 grid, we did not find a cylinder-grid representation without crossings with the ILP. These instances would be potential candidates for an example that cylinder-grid strong embeddability is not equal to single-intersection path-based strong embeddability. Yet, at least at a first examination, there was no obvious way to find a path-based support with single-intersections for these instances. Since we did not implement an exact solution for the test of single-intersection path-based strong embeddability, this inclusion in the hierarchy remains open.

6.2. Case Studies on Real-World Data Sets

Besides the purely randomly generated data from the section above, we applied our method to various real-world datasets. It turned out that in general the two compared clusterings were both cylinder- and plane-grid strongly embeddable. This is probably due to the fact that most clusterings we compared had not more than 10 clusters and were rather similar, resulting in a low grid coverage. In the following we will discuss the results of our methods on two real-world examples.

Figure 6.3 shows a comparison between two clustering methods for graphs. The graph is a network that models co-appearance of characters in the chapters of Victor Hugo's "Les Misérables" [Hug87]. It is part of Knuth's "Stanford GraphBase" (see [Knu93]) and has 77 vertices and 254 edges.

The first clustering method (orange clusters) is based on the *modularity* of the graph, a quality measure for clusterings proposed by Newman (see [New06]). The modularity describes the fraction of edges within clusters of the graph minus the expected fraction of edges in the same clusters of a network with the same node-degrees that would be created at random. We examine the clustering with the maximum modularity as calculated with the ILP approach in [BDG+08].

The second clustering (green clusters) is based on *predominantly connected communities* that have been generated as "overlay clusterings" with the framework described in [HHW13].

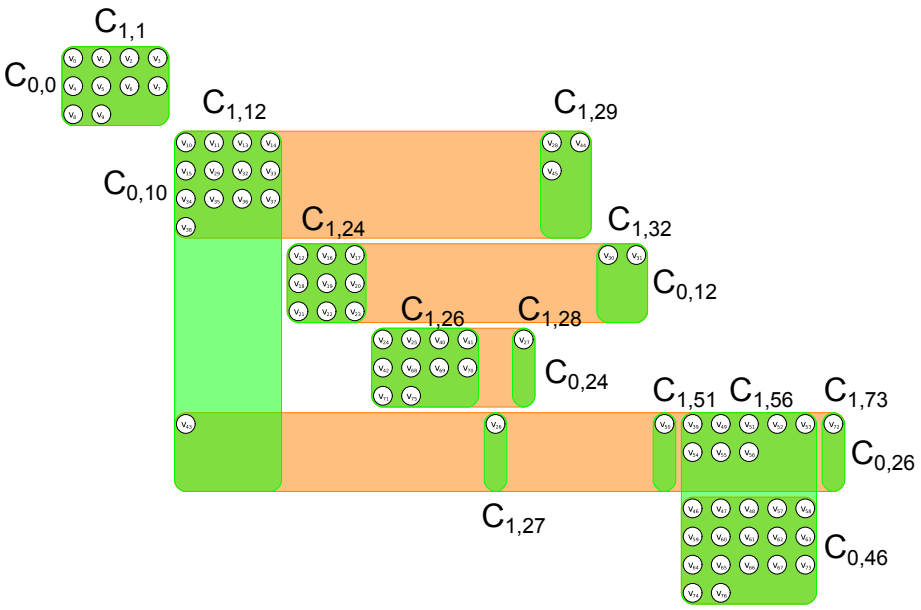


Figure 6.3.: The optimal modularity clustering (orange) compared with the overlay clustering (green) on the "Les Miserables" graph

Since in this example we are only going to do a quantitative analysis, we can ignore the underlying graph structure. Yet for future work on this topic, it would be interesting to evaluate methods of combining the representation of the clusterings with a drawing of the graph.

The plane-grid representation of the two clusterings reveals that both clustering methods set similar cuts between the clusters but have differences in the number of divisions. The 11 clusters of the green clusterings are mostly contained in the six clusters of the orange clustering. The only exceptions are an outlier in $C_{1,12}$ and the aggregation of a major part of $C_{0,26}$ with $C_{0,46}$ in the orange clustering yielding to the big cluster $C_{1,56}$ in the green clustering. The high percentage of overlapping clusters results in a rather low grid-coverage, making it easy to find a grid representation.

With the next example we will show how the graphical representation can help us to evaluate the quality of clustering methods that are not based on graphs. We visualize the results of clustering methods in comparison to the intrinsic clustering of the data. The dataset we used is the "Pen-Based Recognition of Handwritten Digits Data Set" by Alpaydin and Alimoglu that is part of the "UCI Machine Learning Repository" (see [BL13]). It consists of 10992 writing samples of the digits from 0 to 9 from 44 different writers. The samples have been normalized and reduced to eight 2D-coordinates that are spaced equidistantly on the trajectory of the pen. We thus have 16 different parameters (eight times two dimensions). The intrinsic clustering of the data are the digits that were originally drawn.

We clustered the dataset with the WEKA framework (see [HFH+09]) using two different clustering methods that are common choices in many areas of data mining.

The first method we used is k-means. It was first proposed by Steinhaus (see [Ste56]) and is based on the minimization of the sum of squared distances within k clusters. We set the value k to the number of 10 desired clusters (since the intrinsic clustering also contains 10 different digits).

The second method was the more recent DBSCAN (see [EKSX96]) with $\epsilon = 0.45$ and

$minPts = 30$ as parameters. The result were 9 clusters and 657 values that were marked as noise.

The two plane-grid visualizations in Figure 6.4 show the results of the clustering methods compared to the intrinsic clustering of the data.

The orange clusters are in both cases the intrinsic clustering, i.e., the digit that was actually drawn. The green clusters are the result of the clustering algorithm. Since the scale of the generated PDF was too small to distinguish single data points, we highlighted the areas in the figure that contain data points in white.

For the DBSCAN-case, the rightmost green cluster is the aggregation of all data points that were labeled as noise.

The two visualizations allow to evaluate how well the digits were recognized by the two clustering methods. The visualization shows that in general, DBSCAN provided a better approximation to the original clustering. As a simple indicator for this statement, we can look at the grid coverages of the clusterings. The grid coverage is 46 percent in the case of k-means versus 21 percent in the case of DBSCAN without taking into account the noise-cluster and 29 percent with the noise-cluster. An optimal clustering, i.e., a clustering that corresponds to the class code, would have a grid coverage of 10 percent.

Yet the representation conveys a lot more information. We can see how well single digits were recognized and which digits were often mistakingly classified in the same cluster.

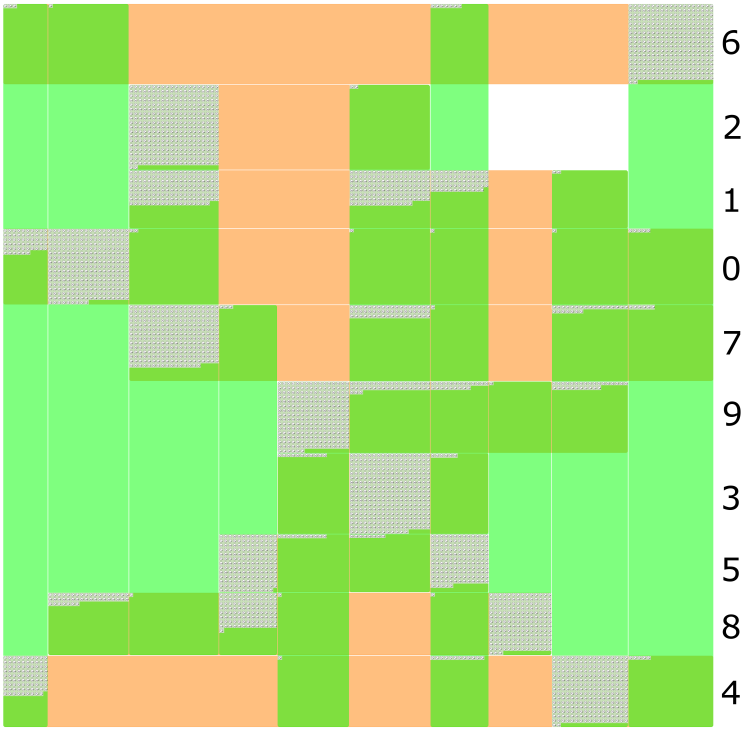
Both algorithms did well on recognizing the digits 6 and 0. In the case of k-means, there are few outliers in both directions, indicating that a few 6s and 0s have been recognized as different digits and that a few different digits have been falsely classified as 6s and 0s.

For DBSCAN, we can even say that the clustering algorithm was able to segment the digits 6, 0 and also 8 without a fault, although in the case of 8, the majority of the data points was dumped as noise. Quite well worked the segmentation for 4, with only a couple instances put in the same cluster with 5s and 9s.

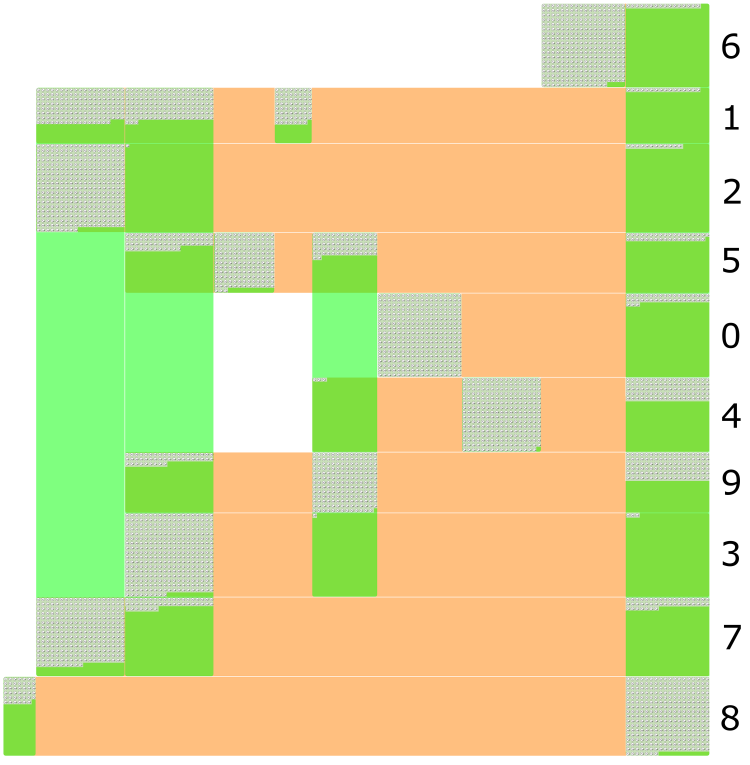
Interesting is also the fact that many 1s, 2s and 7s are put in a common cluster in both cases, probably due to the similarity of their shape. The same is true for 5s and 9s that share a common cluster in the DBSCAN case, while in the k-means clustering, the 5s share a cluster with a big part of the 8s and also with the 1s.

This example shows the general benefits of a simultaneous visualization of two different clusterings. Already our simple drawings of rather experimental character without focus on readability allow us to examine how the clusterings are related to each other. Even for rather large datasets like this one, we can get a good overview at a first glance but can also examine the data down to the level of single data points to examine their belonging to the different clusters.

Yet the example also reveals potential for improvement: The grid representation has many characteristics of a table, and for now, a lot of the drawing area is used up by empty space between or within the clusters. It would be interesting to explore methods of compactifying the drawing in a post-processing step. For large instances like the one we examined, it would also be a nice feature to draw the size of the areas of intersection between two clusters equivalent to the number of common vertices.



(a) k-means (green) compared with the intrinsic clustering (orange)



(b) DBSCAN (green) compared with the intrinsic clustering (orange)

Figure 6.4.: The intrinsic clustering of the "Pen-Digits" data set compared with the results of k-means and DBSCAN

7. Conclusion

In this work we established the theoretical foundations for embedding two (or more) different clusterings and examined practical approaches to generate such embeddings for the case of two clusterings.

We established a hierarchy of eight embeddability classes for two clusterings. For all but one of these classes we could provide examples that show that they are true subsets of their containing class. There are big differences between the classes regarding the complexity of deciding whether two clusterings belong to the class or not.

The simplest form of an embedding, a weak embedding, exists always. The strictest class of embeddability, full embeddability, can be checked in linear time. Regarding the classes in between, we could show $\mathcal{NP}$-completeness for the central concept of strong embeddability. For the two classes of grid embeddings in the plane and on a cylinder, $\mathcal{NP}$-completeness is implied by the equivalence to grid-intersection graphs. For the remaining three classes, the complexity of the decision problem is still unknown.

An implicit result of our proof of $\mathcal{NP}$-completeness for strong embeddability is the $\mathcal{NP}$-completeness of deciding vertex planarity for 2-regular hypergraphs.

Despite the hardness of some of the problems, our practical results allow us to generate embeddings of two clusterings for the grid-based classes of embeddability. The focus on grid representations provides a simple way to describe and generate embeddings. We have the choice between an exact but potentially slow solution with an integer linear program or a fast approximation with a heuristical approach which is preferrable for large instances.

By minimizing crossings between disjoint clusters in the grid, we try to find a strong embedding and otherwise provide a weak embedding with a low number of such crossings. Our experiments suggest that the embeddability mainly depends on the similarity of the clusterings as expressed by their grid coverage. This means that for typical real-world instances with a low grid coverage, the chances of finding a strong embedding are quite good.

A by-product of the integer linear programs is an ILP-approach that decides whether a graph is a grid intersection graph in the plane, on the annulus or (with a slight modification) on the torus.

7.1. Open Problems in the Hierarchy

The main question that is still open in the hierarchy is whether single-intersection path-based strong embeddability is equivalent to cylinder-grid strong embeddability or not. It seems obvious that there exist more possible single-intersection strong embeddings for a given instance than cylinder-grid strong embeddings. Yet we did not find a counterexample that proves cylinder-grid strong embeddability to be a proper subset of single-intersection path-based strong embeddability.

Another open problem is the complexity of deciding whether there exists a single-intersection, a path-based or a single-intersection path-based strong embedding. Our proof for the $\mathcal{NP}$-completeness of strong embeddability testing in general uses non-path-based clusters and multiple intersections between clusters and is therefore not applicable.

7.2. Future Work

In order to use our theoretical results for the generation of well-readable graphical representations based on embeddings, the next step would be to optimize some aesthetical criteria of the output. One approach to increase readability could be to limit the extent of clusters and to draw the curves in convex shapes.

Since clusterings often occur in graphs and networks, it would be nice to visualize the clusterings in combination with the network. This could be realized with a force-based approach that layouts the subgraph induced by equivalent vertices within the clusters. In parallel, clusters could be placed according to forces of edges between their contained nodes while respecting the embedding.

In our current plane-grid representations, layouts with low grid coverages take up a lot more space than necessary. It would be interesting to investigate methods for making the layouts more compact. This could be achieved in a post-processing step. Especially if we would allow multiple clusters to be placed in the same row or column if they do not overlap horizontally or vertically, we could optimize the space used for the representation.

Another interesting feature for grid representations would be to fix the order of one of the two clusterings. This could be used if the fixed clustering is an ordered set and we want to keep that order in the grid for the clarity of the representation.

In terms of the heuristic we developed, there is certainly a lot of room for improvement. Since we start with only one ordering and then follow a greedy approach, it is quite likely to get stuck in a local minimum for the crossing number. Ideas for improvement of the method would be to start multiple runs with different initial orderings and additionally allow row- or column-swaps that generate a higher crossing number with a certain probability. This approach would be based on the ideas of *Simulated Annealing* (see [KGV83]).

Last but not least, a next step could be to extend the work to families of three or more clusterings, both from the theoretical and practical point of view. It would be an interesting challenge to generate readable drawings of more than two clusters.

Appendix

A. File Formats

The ascii-file format we use to describe a clustering is simple: Every line represents a vertex and contains an integer that represents the cluster-ID of the vertex. Figure A.1 gives an example of two clusterings on a set of twenty vertices.

4	3
4	5
0	5
0	0
4	5
2	7
0	6
3	4
2	8
1	2
2	3
0	8
1	1
4	5
3	5
4	6
1	7
4	6
4	4
4	0
(a) clustering0.txt	(b) clustering1.txt

Figure A.1.: The input files for the two clusterings shown in Fig. 5.1

B. Calculating the Number of Bad Crossings in Grid Representations

When we calculate the number of bad crossings, we use a check-matrix representing the grid. The value in the matrix is set to 0 if no ribbon covers the cell, to 1 if one ribbon covers the cell and to 2 if two ribbons intersect in the cell. We then just count the number of empty grid cells with value 2 in the check matrix. The functions *firstNonEmpty()* and *lastNonEmpty()* return the first or last non-empty grid cell in the given row or column.

The following algorithm calculates the number of bad crossings in a plane-grid representation:

Algorithm 2: currentCrossingNumPlane()

Input : grid representation G of two clusterings
Output: number of bad crossings in G when viewed as plane-grid

```
numCrossings ← 0
generate check matrix M
initialize M with 0s
foreach row i in G do
    foreach cell j in row i do
        if firstNonEmpty(i) ≤ j ≤ lastNonEmpty(i) then
            M(i, j) ← M(i, j) + 1
foreach column j in G do
    foreach cell i in column j do
        if firstNonEmpty(j) ≤ i ≤ lastNonEmpty(j) then
            M(i, j) ← M(i, j) + 1
foreach cell (i, j) in G do
    if M(i, j) = 2 AND isEmpty(G(i, j)) then
        numCrossings ← numCrossings +1
```

For cylinder-grid representations, we use the following, slightly more complicated algorithm. We calculate the number of bad crossings for the cases that rows or columns wrap around the cylinder separately and then choose the case that induces the lowest number of bad crossings. The function *findBiggestBreak()* finds the largest connected section of grid cells that would cause bad crossings if the ribbon of the cluster would cross this section. This is where we set the cut for the ribbon. The cylindrical structure can lead to the value *firstNonEmpty()* being higher than *lastNonEmpty()*, indicating that the row or column is wrapped. Algorithm 3 shows the process.

Algorithm 3: currentCrossingNumCylinder()

Input : grid representation G of two clusterings
Output: number of bad crossings in G when viewed as cylinder-grid

```
numCrossingsRowWrap ← 0
numCrossingsColWrap ← 0
generate check matrix M_r
generate check matrix M_c
initialize M_r and M_c with 0s
foreach column j in G do
    foreach cell i in column j do
        if firstNonEmpty(j) ≤ i ≤ lastNonEmpty(j) then
            M_r(i, j) ← M_r(i, j) + 1
foreach row i in G do
    firstNonEmpty(i) = findBiggestBreak(i).end
    lastNonEmpty(i) = findBiggestBreak(i).start
    foreach cell j in row i do
        if firstNonEmpty(i) < lastNonEmpty(i) then
            if firstNonEmpty(i) ≤ j ≤ lastNonEmpty(i) then
                M_r(i, j) ← M_r(i, j) + 1
        else
            if firstNonEmpty(i) ≤ j OR j ≤ lastNonEmpty(i) then
                M_r(i, j) ← M_r(i, j) + 1
foreach cell (i, j) in G do
    if M_r(i, j) = 2 AND isEmpty(G(i, j)) then
        numCrossingsRowWrap ← numCrossingsRowWrap +1
foreach row i in G do
    foreach cell j in row i do
        if firstNonEmpty(i) ≤ j ≤ lastNonEmpty(i) then
            M_c(i, j) ← M_c(i, j) + 1
foreach column j in G do
    firstNonEmpty(j) = findBiggestBreak(j).end
    lastNonEmpty(j) = findBiggestBreak(j).start
    foreach cell i in column j do
        if firstNonEmpty(j) < lastNonEmpty(j) then
            if firstNonEmpty(j) ≤ i ≤ lastNonEmpty(j) then
                M_c(i, j) ← M_c(i, j) + 1
        else
            if firstNonEmpty(j) ≤ i OR i ≤ lastNonEmpty(j) then
                M_c(i, j) ← M_c(i, j) + 1
foreach cell (i, j) in G do
    if M_c(i, j) = 2 AND isEmpty(G(i, j)) then
        numCrossingsColWrap ← numCrossingsColWrap +1
return max(numCrossingsRowWrap, numCrossingsColWrap)
```

C. Random Clustering Generators

The following two algorithms show how we generated clusterings at random for experiments. Algorithm 4 takes a number of vertices and adds them randomly to clusters of the two clusterings. Algorithm 5 takes a desired grid-coverage as input and then adds exactly one common vertex to random pairs of clusters until the grid-coverage is reached.

Algorithm 4: randomClusteringsNumVertices()

Input : number of vertices n,
number of clusters in clusterings $numClusters1$, $numClusters2$
Output: two random cluster-files $file1$, $file2$
while $n > 0$ **do**
    $file1$.addRandomNumber(0, $numClusters1 - 1$)
    $file2$.addRandomNumber(0, $numClusters2 - 1$)
    $n \leftarrow n - 1$

Algorithm 5: randomClusteringsGridCoverage()

Input : grid coverage c,
number of clusters in clusterings $numClusters1$, $numClusters2$
Output: two random cluster-files $file1$, $file2$
generate check matrix M of size $numClusters1 \times numClusters2$
initialize M with 0's
numVertices $\leftarrow c \cdot numClusters1 \cdot numClusters2$
while $numVertices > 0$ **do**
    repeat
        rand1 $\leftarrow$ randomNumber(0, $numClusters1$)
        rand2 $\leftarrow$ randomNumber(0, $numClusters2$)
    until $M(rand1, rand2) = 0$
    $file1$.add(rand1)
    $file2$.add(rand2)
    $M(rand1, rand2) \leftarrow 1$
    $numVertices \leftarrow numVertices - 1$

D. Experimental Results

The following tables show the raw data from our experiments regarding the quality of the heuristic for plane-grid representations on the 10×10 grid. Table D.1 shows the best crossing number found by the heuristic for the twenty test instances of each grid-coverage, Table D.2 the exact solution found by the integer linear program. The running times of the ILP for the instances on our test machine are listed in Table D.3.

	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
#01	0	0	1	1	1	1	2	1	2	2	0	2	1	1
#02	0	0	1	0	0	2	5	1	0	1	2	0	2	1
#03	0	0	0	1	2	4	2	2	3	1	1	2	0	0
#04	0	0	0	0	3	3	2	1	1	1	0	1	0	1
#05	0	0	1	0	1	2	2	2	3	1	2	1	0	0
#06	0	0	1	0	4	3	2	4	2	2	1	2	0	0
#07	0	0	1	0	1	2	1	4	1	3	3	1	0	0
#08	0	0	0	0	1	2	2	1	3	1	0	2	0	1
#09	0	0	1	1	0	4	2	3	2	1	3	1	0	0
#10	0	0	1	0	2	2	2	1	0	1	2	0	1	1
#11	0	0	0	0	1	1	3	1	2	3	1	2	1	0
#12	0	0	0	1	1	5	2	2	1	1	0	2	1	0
#13	0	0	0	0	2	2	3	2	1	2	0	0	1	0
#14	0	0	0	2	1	4	3	2	0	2	2	0	2	0
#15	0	0	1	1	3	0	2	1	0	1	1	1	1	1
#16	0	0	1	0	1	3	1	3	3	3	1	3	0	0
#17	0	0	0	1	1	2	1	2	0	2	1	0	1	0
#18	0	0	0	2	1	3	2	3	0	1	1	2	0	0
#19	0	1	1	1	1	0	3	2	3	2	0	1	0	0
#20	0	0	1	0	1	3	2	1	2	2	2	2	0	0
Average	0	0.05	0.55	0.55	1.4	2.4	2.2	1.95	1.45	1.65	1.15	1.25	0.55	0.3

Table D.1.: Crossing numbers in plane-grid representations calculated by the heuristic

	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
#01	0	0	1	0	0	1	1	1	1	2	0	2	1	1
#02	0	0	0	0	0	1	2	1	0	1	2	0	2	1
#03	0	0	0	1	2	2	1	1	3	1	1	2	0	0
#04	0	0	0	0	1	1	0	1	0	1	0	1	0	1
#05	0	0	0	0	0	1	0	2	3	1	2	1	0	0
#06	0	0	0	0	1	2	1	3	2	2	1	2	0	0
#07	0	0	0	0	0	1	1	1	1	3	3	1	0	0
#08	0	0	0	0	0	2	0	0	3	1	0	2	0	1
#09	0	0	0	0	0	3	2	2	1	1	3	1	0	0
#10	0	0	1	0	1	1	2	1	0	1	1	0	1	1
#11	0	0	0	0	0	1	2	1	1	3	1	1	1	0
#12	0	0	0	1	0	3	2	2	1	0	0	2	1	0
#13	0	0	0	0	2	0	2	1	1	2	0	0	1	0
#14	0	0	0	1	0	2	3	1	0	0	2	0	2	0
#15	0	0	1	1	1	0	1	0	0	0	1	1	1	1
#16	0	0	0	0	0	2	1	1	1	2	1	3	0	0
#17	0	0	0	1	0	1	1	2	0	1	1	0	1	0
#18	0	0	0	2	0	2	1	2	0	0	1	2	0	0
#19	0	0	0	0	0	0	2	2	2	1	0	1	0	0
#20	0	0	0	0	0	1	2	1	2	2	1	2	0	0
Average	0	0	0.15	0.35	0.4	1.35	1.35	1.3	1.1	1.25	1.05	1.2	0.55	0.3

Table D.2.: Exact crossing numbers in plane-grid representations calculated by the integer linear program

	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
#01	0.19	0.13	28.15	23.64	4.10	62.34	9.87	5.91	8.21	11.02	0.97	5.19	46.99	13.70
#02	0.46	0.40	5.52	0.48	1.71	24.88	32.89	8.69	1.95	4.32	7.89	0.49	4.50	5.35
#03	0.09	1.91	0.48	37.56	49.45	69.18	5.11	5.99	17.86	2.62	2.91	5.83	0.58	0.02
#04	0.09	0.47	1.94	2.49	29.96	16.40	19.83	11.99	1.09	3.46	0.18	27.52	0.03	21.39
#05	0.51	1.39	12.83	1.40	2.65	13.28	8.43	20.70	17.22	1.79	8.19	2.61	0.08	0.03
#06	0.32	5.75	2.78	5.32	29.10	20.88	13.87	42.43	9.60	11.46	3.05	4.82	0.03	0.01
#07	0.14	4.14	1.78	0.47	3.08	8.46	8.22	13.56	7.36	11.86	10.93	6448.73	0.02	0.04
#08	0.50	1.01	5.09	2.98	9.17	28.59	5.36	0.73	22.79	2.56	2.21	4.69	0.10	9.37
#09	0.14	1.09	4.70	2.58	2.30	35.65	25.63	15.19	7.07	4.88	9.41	422.70	0.10	0.04
#10	4.54	0.64	59.98	2.61	15.53	40.78	15.39	5.72	1.60	3.66	3.68	1.17	23.04	8.86
#11	0.20	2.59	0.55	1.97	3.19	10.21	22.56	7.64	22.43	13.02	2.81	2.94	122.66	0.02
#12	0.71	3.12	2.60	15.16	11.70	73.27	17.94	20.95	97.15	1.45	0.18	6.28	91.30	0.04
#13	0.67	3.06	1.79	1.23	78.06	1.88	27.27	15.71	5.41	10.66	1.09	0.09	2309.23	0.01
#14	2.95	3.96	4.02	27.31	2.16	38.84	63.81	6.64	4.01	3.64	5.88	0.31	1.86	0.02
#15	0.50	1.43	29.68	343.09	29.91	3.74	13.16	2.65	1.14	0.49	2.65	244.90	1.30	5.78
#16	0.63	1.46	3.12	4.30	4.13	44.64	8.52	7.21	5.98	14.61	5.10	3.65	0.08	0.05
#17	0.06	3.26	0.23	21.13	6.67	11.43	8.55	13.03	2.12	3.39	3.45	0.17	1993.75	0.04
#18	0.49	0.48	4.07	86.80	3.02	20.61	10.76	18.36	1.83	1.23	3.65	3.53	0.08	0.04
#19	0.15	5.15	6.19	13.64	10.28	5.42	24.19	28.55	19.99	96.74	0.71	289.44	0.02	0.02
#20	0.61	1.78	11.00	3.58	5.53	9.00	29.54	5.48	6.08	7.86	2.13	3.84	0.12	0.02
Average	0.70	2.16	9.33	29.89	15.09	26.97	18.55	12.86	13.04	10.54	3.85	373.95	229.79	3.24

Table D.3.: Running times of the integer linear program on our test machine

Bibliography

[ARRC11] B. Alper, N. H. Riche, G. Ramos, and M Czerwinski: *Design study of linesets, a novel set visualization technique.* IEEE Transactions on Visualization and Computer Graphics, 17(12):2259–2267, 2011. http://dx.doi.org/10.1109/TVCG.2011.186.

[BCPS11a] U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry: *Blocks of hypergraphs: applied to hypergraphs and outerplanarity.* In *Proceedings of the 21st international conference on Combinatorial Algorithms*, IWOCA'10, pages 201–211, 2011. http://dx.doi.org/10.1007/978-3-642-19222-7_21.

[BCPS11b] U. Brandes, S. Cornelsen, B. Pampel, and A. Sallaberry: *Path-based supports for hypergraphs.* In *Proceedings of the 21st international conference on Combinatorial Algorithms*, IWOCA'10, pages 20–33, 2011. http://dx.doi.org/10.1007/978-3-642-19222-7_3.

[BDG+08] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner: *On modularity clustering.* IEEE Transactions on Knowledge and Data Engineering, 20(2):172–188, 2008. http://dx.doi.org/10.1109/TKDE.2007.190689.

[Ben59] S. Benzer: *On the topology of the genetic fine structure.* Proceedings of the National Academy of Sciences of the United States of America, 45(11):1607–1620, 1959. http://dx.doi.org/10.1073/pnas.45.11.1607.

[Ber89] C. Berge: *Hypergraphs: Combinatorics of Finite Sets.* North-Holland Mathematical Library. North Holland, 1989. http://books.google.de/books?id=jEyfse-EKf8C.

[BL13] K. Bache and M. Lichman: *UCI machine learning repository*, 2013. http://archive.ics.uci.edu/ml.

[BvKM+10] K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek: *On planar supports for hypergraphs.* In *Proceedings of the 17th international conference on Graph Drawing*, GD'09, pages 345–356, 2010. http://dx.doi.org/10.1007/978-3-642-11805-0_33.

[Cay89] A. Cayley: *A theorem on trees.* Quarterly Journal of Mathematics, 23:376–378, 1889. http://dx.doi.org/10.1017/CBO9780511703799.010.

[CG09] J. Chalopin and D. Gonçalves: *Every planar graph is the intersection graph of segments in the plane: extended abstract.* In *Proceedings of the 41st annual ACM symposium on Theory of Computing*, STOC '09, pages 631–638, 2009. http://dx.doi.org/10.1145/1536414.1536500.

[CGJ+13] M. Chimani, C. Gutwenger, M. Jünger, K. Klein, P. Mutzel, and M. Schulz.: *The open graph drawing framework*, 2013. http://www.ogdf.net/.

[CGO10] J. Chalopin, D. Gonçalves, and P. Ochem: *Planar graphs have 1-string representations.* Discrete & Computational Geometry, 43:626–647, 2010. http://dx.doi.org/10.1007/s00454-009-9196-9.

[Cho07] S. C. Chow: *Generating and drawing area-proportional Euler and Venn diagrams.* PhD thesis, University of Victoria, 2007. http://portal.acm.org/citation.cfm?id=1414778.

[CJKV12] S. Chaplick, V. Jelínek, J. Kratochvíl, and T. Vyskocil: *Bend-bounded path intersection graphs: Sausages, noodles, and waffles on a grill.* Graph-Theoretic Concepts in Computer Science, 7551:274–285, 2012. http://dx.doi.org/10.1007/978-3-642-34611-8_28.

[CPC09] C. Collins, G. Penn, and S. Carpendale: *Bubble sets: Revealing set relations with isocontours over existing visualizations.* IEEE Transactions on Visualization and Computer Graphics, 15(6):1009–1016, 2009. http://dx.doi.org/10.1109/TVCG.2009.122.

[CR04] S. C. Chow and F. Ruskey: *Drawing area-proportional venn and euler diagrams.* In *Proceedings of the 11th international conference on Graph Drawing*, volume 2912 of *GD'03*, pages 466–477, 2004. http://dx.doi.org/10.1007/978-3-540-24595-7_44.

[dBK09] M. de Berg and A. Khosravi: *Finding perfect auto-partitions is $\mathcal{NP}$-hard.* In *Abstracts of the 25th European Workshop on Computational Geometry*, pages 255–258, 2009. http://www.win.tue.nl/~Akhosrav/papers/auto-part-EWCG.pdf.

[DCS06] N. Durand, B. Crémilleux, and E. Suzuki: *Visualizing transactional data with multiple clusterings for knowledge discovery.* In *Foundations of Intelligent Systems*, volume 4203 of *Lecture Notes in Computer Science*, pages 47–57. 2006. http://dx.doi.org/10.1007/11875604_7.

[EdCCC42] L. Euler, J. A. N. de Caritat Condorcet, and A. A. Cournot: *Lettres de L. Euler à une princesse d'Allemagne sur divers sujets de physique et de philosophie.* Number 1 in *Lettres de L. Euler à une princesse d'Allemagne sur divers sujets de physique et de philosophie.* Hachette, 1842. http://books.google.de/books?id=1QsLAAAAMAAJ.

[EET76] G. Ehrlich, S. Even, and R. E. Tarjan: *Intersection graphs of curves in the plane.* Journal of Combinatorial Theory, Series B, 21(1):8 – 20, 1976. http://dx.doi.org/10.1016/0095-8956(76)90022-8.

[EF97] P. Eades and Q. W. Feng: *Multilevel visualization of clustered graphs.* In *Proceedings of the Symposium on Graph Drawing*, GD '96, pages 101–112, 1997. http://dx.doi.org/10.1007/3-540-62495-3_41.

[EKSX96] M. Ester, H. P. Kriegel, J. Sander, and X. Xu: *A density-based algorithm for discovering clusters in large spatial databases with noise.* In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD-96, pages 226–231, 1996. http://www.aaai.org/Library/KDD/1996/kdd96-037.php.

[FCE95] Q. W. Feng, R. F. Cohen, and P. Eades: *Planarity for clustered graphs.* In *Algorithms - ESA '95*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. 1995. http://dx.doi.org/10.1007/3-540-60313-1_145.

[FS58] R. O. Ferguson and L. F. Sargent: *Linear programming.* McGraw-Hill, 1958. http://books.google.de/books?id=4RlVAAAAMAAJ.

[GHK99] J. Gil, J. Howse, and S. Kent: *Formalizing spider diagrams.* In *Proceedings of the IEEE Symposium on Visual Languages*, pages 130–137, 1999. http://dx.doi.org/10.1109/VL.1999.795884.

[GJ79] M. R. Garey and D. S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., 1979. http://books.google.de/books?id=fjxGAQAAIAAJ.

[GJ83] M. R. Garey and D. S. Johnson: *Crossing number is $\mathcal{NP}$-complete.* SIAM Journal on Algebraic and Discrete Methods, 4(3):312–316, 1983. http://dx.doi.org/10.1137/0604033.

[GNS09] I. Griva, S. Nash, and A. Sofer: *Linear and Nonliner Optimization, 2nd Edition.* Society for Industrial and Applied Mathematics, 2009. http://books.google.de/books?id=uOJ-Vg1BnKgC.

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten: *The weka data mining software: an update.* SIGKDD Explorations Newsletter, 11(1):10–18, 2009. http://doi.acm.org/10.1145/1656274.1656278.

[HHW13] M. Hamann, T. Hartmann, and D. Wagner: *Hierarchies of predominantly connected communities.* In *Proceedings of the 13th International Symposium on Algorithms and Data Structures (WADS'13)*, Lecture Notes in Computer Science, 2013. http://arxiv.org/abs/1305.0757, (to appear).

[HNZ91] I. B. A. Hartman, I. Newman, and R. Ziv: *On grid intersection graphs.* Discrete Mathematics, 87(1):41–52, 1991. http://dx.doi.org/10.1016/0012-365X(91)90069-E.

[HT74] J. Hopcroft and R. E. Tarjan: *Efficient planarity testing.* Journal of the ACM (JACM), 21(4):549–568, 1974. http://dx.doi.org/10.1145/321850.321852.

[Hug87] V. Hugo: *Les Misérables.* Les Misérables. T.Y. Crowell & Company, 1887. http://books.google.de/books?id=XdwPAAAAYAAJ.

[HY00] D. Harel and G. Yashchin: *An algorithm for blob hierarchy layout.* In *Proceedings of the working conference on Advanced visual interfaces*, AVI '00, pages 29–40, 2000. http://dx.doi.org/10.1145/345513.345240.

[Inc13] Gurobi Optimization Inc.: *Gurobi optimizer reference manual*, 2013. http://www.gurobi.com.

[Jor87] C. Jordan: *Cours d'analyse de l'École polytechnique.* Number 3 in *Cours d'analyse de l'École polytechnique.* Gauthier-Villars et fils, 1887. http://books.google.de/books?id=cx0PAAAAIAAJ.

[JP87] D. S. Johnson and H. O. Pollak: *Hypergraph planarity and the complexity of drawing venn diagrams.* Journal of Graph Theory, 11(3):309–325, 1987. http://dx.doi.org/10.1002/jgt.3190110306.

[Kar72] R. M. Karp: *Reducibility among combinatorial problems.* In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. http://dx.doi.org/10.1007/978-3-540-68279-0_8.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi: *Optimization by simulated annealing.* Science, 220(4598):671–680, 1983. http://dx.doi.org/10.1126/science.220.4598.671.

[Knu93] D. E. Knuth: *The Stanford GraphBase: a platform for combinatorial computing*. ACM, 1993. http://books.google.de/books?id=BdTUPQAACAAJ.

[KP96] J. Kratochvíl and T. M. Przytycka: *Grid intersection and box intersection graphs on surfaces (extended abstract)*. In *Proceedings of the Symposium on Graph Drawing*, GD '95, pages 365–372, 1996. http://dx.doi.org/10.1007/BFb0021820.

[KR92] D. E. Knuth and A. Raghunathan: *The problem of compatible representatives*. SIAM Journal on Discrete Mathematics, 5(3):422–427, 1992. http://dx.doi.org/10.1137/0405033.

[Kra94] J. Kratochvíl: *A special planar satisfiability problem and a consequence of its $\mathcal{NP}$-completeness*. Discrete Applied Mathematics, 52(3):233–252, 1994. http://dx.doi.org/10.1016/0166-218X(94)90143-0.

[KvKS09] M. Kaufmann, M. van Kreveld, and B. Speckmann: *Subdivision drawings of hypergraphs*. In *Graph Drawing*, pages 396–407. 2009. http://dx.doi.org/10.1007/978-3-642-00219-9_39.

[Lic82] D. Lichtenstein: *Planar formulae and their uses*. SIAM Journal on Computing, 11(2):329–343, 1982. http://dx.doi.org/10.1137/0211025.

[Mäk90] E. Mäkinen: *How to draw a hypergraph*. International Journal of Computer Mathematics, 34(3-4):177–185, 1990. http://dx.doi.org/10.1080/00207169008803875.

[MM99] T. A. McKee and F. R. McMorris: *Topics in Intersection Graph Theory*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, 1999. http://books.google.de/books?id=2E10MLGRjFUC.

[NC88] T. Nishizeki and N. Chiba: *Planar Graphs: Theory and Algorithms*. North-Holland Mathematics Studies. Elsevier Science, 1988. http://books.google.de/books?id=clMoNmX-x6MC.

[New06] M. E. J. Newman: *Modularity and community structure in networks*. Proceedings of the National Academy of Sciences, 103(23):8577–8582, 2006. http://dx.doi.org/10.1073/pnas.0601602103.

[Pap81] C. H. Papadimitriou: *On the complexity of integer programming*. Journal of the ACM, 28(4):765–768, 1981. http://dx.doi.org/10.1145/322276.322287.

[PT02] J. Pach and G. Tóth: *Recognizing string graphs is decidable*, 2002. http://dx.doi.org/10.1007/3-540-45848-4_20.

[Ran71] W. M. Rand: *Objective criteria for the evaluation of clustering methods*. Journal of the American Statistical Association, 66(336):846–850, 1971. http://dx.doi.org/10.1080/01621459.1971.10482356.

[RS69] F. S. Roberts and J. H. Spencer: *A Characterization of Clique Graphs*. Memorandum (Rand Corporation). Rand Corporation, 1969. http://books.google.fr/books?id=tRpvGQAACAAJ.

[SAA09] P. Simonetto, D. Auber, and D. Archambault: *Fully automatic visualisation of overlapping sets*. In *Proceedings of the 11th Eurographics / IEEE - VGTC conference on Visualization*, EuroVis'09, pages 967–974, 2009. http://dx.doi.org/10.1111/j.1467-8659.2009.01452.x.

[Sch84] E. R. Scheinerman: *Intersection Classes and Multiple Intersection Parameters of Graphs*. Princeton University, 1984. http://books.google.de/books?id=KpQEGwAACAAJ.

[Sin66] F. W. Sinden: *Topology of thin film RC-circuits*. Bell System Technical Journal, 45:1639–1662, 1966. http://archive.org/details/bstj45-9-1639.

[SSu03] M. Schaefer, E. Sedgwick, and D. Štefankovič: *Recognizing string graphs in $\mathcal{NP}$*. Journal of Computer and System Sciences, 67(2):365–380, 2003. http://dx.doi.org/10.1016/S0022-0000(03)00045-X.

[Ste56] H. Steinhaus: *Sur la division des corps matériels en parties*. Bulletin of the Polish Academy of Sciences, Cl. III. 4, pages 801–804, 1956.

[Su04] M. Schaefer and D. Štefankovič: *Decidability of string graphs*. Journal of Computer and System Sciences, 68(2):319–334, 2004. http://dx.doi.org/10.1016/j.jcss.2003.07.002.

[VBV04] A. Verroust-Blondet and M. L. Viaud: *Results on hypergraph planarity*, 2004. http://hal.inria.fr/inria-00389591, (Unpublished manuscript).

[Ven80] J. Venn: *On the diagrammatic and mechanical representation of propositions and reasonings*. Philosophical Magazine Series 5, 10(59):1–18, 1880. http://dx.doi.org/10.1080/14786448008626877.

[Wal75] T. R. S. Walsh: *Hypermaps versus bipartite maps*. Journal of Combinatorial Theory, Series B, 18(2):155–163, 1975. http://dx.doi.org/10.1016/0095-8956(75)90042-8.

[Wil96] D. B. Wilson: *Generating random spanning trees more quickly than the cover time*. In *Proceedings of the 28th annual ACM symposium on Theory of Computing*, STOC '96, pages 296–303, 1996. http://dx.doi.org/10.1145/237814.237880.

[Zyk74] A. A. Zykov: *Hypergraphs*. Russian Mathematical Surveys, 29(6):89–156, 1974. http://dx.doi.org/10.1070/RM1974v029n06ABEH001303.