Constrained Planarity and Augmentation Problems

Inaugural-Dissertation
zur
Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Universität zu Köln

vorgelegt von
Merijam Percan
aus Lübeck

Köln 2007

Berichterstatter:	**Prof. Dr. Michael Jünger**
	HD. Dr. Hubert Randerath
Tag der mündlichen Prüfung:	**18. Januar 2008**

Acknowledgments

I would like to thank everyone who supported me in writing this thesis. First of all, I would like to thank my supervisor Prof. Dr. Michael Jünger for giving me the possibility to do research in automatic graph drawing and graph theory and for providing much help during internal seminars.

I am grateful for all the constructive and helpful discussions during cooperations with Dr. Christoph Buchheim, Dr. Elisabeth Gassner, Carsten Gutwenger, Prof. Dr. Michael Jünger, Dr. Sebastian Leipert, Annette Menze, Prof. Dr. Petra Mutzel, Dr. Markus Schaefer, Prof. Dr. Klaus Truemper and Dr. René Weiskircher.

Especially, I would like to thank Klaus Truemper for bringing me into the research field of graph theory, for all his help and great ideas that influence significantly the second part of my thesis. I am very grateful that I got the possibility to work with him during a two months stay at the UT Dallas.

A lot of thanks go to Thomas Lange, Marc Egger, Holger Flier, Constantin Hellweg, Annette Menze, Robin Noack, Ramin Sahamie and Marc Sprenger for the huge support in technical questions.

Furthermore, I would like to thank Michael Belling and Ursula Neugebauer for helping me in the acquisition of literature and all administrative processes.

A big thank goes also to Frank Baumann, Matthias Elf, Dr. Diana Fanghänel, Dr. Stefan Hachul, Dr. Frauke Liers, Gregor Pardella, HD Dr. Bert Randerath, Michael Schulz, Andrea Wagner and all mentioned before for their open doors.

Further, I would like to thank my family, for supporting me in all aspects in my life. I am very grateful for their help and for giving me the possibility to study. Many thanks go to Dr. Katrina Riehl and Ingrid Truemper for their language support and friendship. Last but not least, a big thank is addressed to Dr. Martin Gotzes and his family.

ABSTRACT

This thesis has three parts:

Part I focus on clustered graphs. A clustered graph $C = (G, T)$ consists of an undirected graph G and a rooted tree T in which the leaves of T correspond to the vertices of $G = (V, E)$. Each vertex μ in T corresponds to a subset of the vertices of the graph called "cluster". c-planarity is a natural extension of graph planarity for clustered graphs, and plays an important role in automatic graph drawing. The complexity status of c-planarity testing is unknown. It has been shown in [30, 18, 20] that c-planarity can be tested in linear time for c-connected graphs, i.e., graphs in which the cluster induced subgraphs are connected.

In this thesis, we provide a polynomial time algorithm for c-planarity testing of "almost" c-connected clustered graphs, i.e., graphs for which all nodes corresponding to the non-c-connected clusters lie on the same path in T starting at the root of T, or graphs in which for each non-connected cluster its super-cluster and all its siblings in T are connected; see [40] and Sections 7.3 to 7.5 of this thesis. Additionally, we provide a polynomial time algorithm for planar clustered graphs for which $G - G(\nu)$ is connected for each cluster ν in Section 7.2. The algorithms are based on the concepts for the subgraph induced planar connectivity augmentation problem presented in [41] and Chapter 6 of this thesis.

Given a planar graph $G = (V, E)$ and a vertex set $W \subseteq V$, the subgraph induced planar connectivity augmentation problem asks for a minimum cardinality set F of additional edges with end vertices in W such that $G' = (V, E \cup F)$ is planar and the subgraph of G' induced by W is connected. We describe a linear time algorithm based on SPQR-trees that tests if a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set.

Furthermore, we give some characterization for c-planar clustered graphs based on dual graphs and forbidden minors in Chapter 4 and Section 7.1.

In Chapter 8, we provide a linear time algorithm for augmenting c-connected c-planar embedded clustered graphs $C = (G, T)$ by adding edges such that C remains c-planar embedded even after applying any planar graph augmentation algorithm afterwards.

Parts II deals with edge deletion and bimodal crossing minimization. In Chapter 9 we consider edge deletion problems. In Section 9.1 we prove that the maximum planar subgraph problem remains $\mathcal{NP}$-complete even for non-planar graphs without a minor isomorphic

to either K_5 or $K_{3,3}$, respectively. In Section 9.2, we investigate the problem of finding a minimum weighted set of edges whose removal results in a graph without minors that are contractible onto a prespecified set of vertices. Such minors are called rooted. The problem of a minimum weighted deletion of all rooted $K_{i,3}$-minors for a fixed $i \in \mathbb{N}$ is proved to be $\mathcal{NP}$-hard on general graphs. Furthermore, an $O(n^3)$ time algorithm is developed for the rooted $K_{1,3}$-minor deletion problems on planar graphs while for the rooted $K_{2,3}$-free minor planar graphs a characterization is presented.

In Chapter 10 we consider the problem of drawing a directed graph in two dimensions with a minimal number of crossings such that for every node the incoming and outgoing edges are separated consecutively in the cyclic adjacency lists. We adapt the usual planarization method and other approaches in a simple way. We report experimental results for the increase in the number of crossings involved by this additional restriction on the set of feasible drawings. We can summarize the statement of this chapter as follows: whenever the direction of edges in a graph carries significant information, this should be stressed by separating incoming and outgoing edges in the adjacency lists. We show how crossing reduction algorithms can be adapted in order to comply with this requirement. The necessary changes are not only easy to implement but also neutral with respect to runtime. As our experiments show, the number of crossings can be expected to grow only slightly for practical instances.

Part III summarizes, concludes and gives some ideas for future work on the complexity status of c-planarity studied in Part I of this thesis.

ZUSAMMENFASSUNG

Der erste Teil dieser Arbeit beschäftigt sich mit einer speziellen Planarität definiert auf einer speziellen Graphen-Klasse. Zuerst möchten wir uns mit Definitionen von Wendy Feng und Peter Eades (s. [29, 30]) beschäftigen, die wir für unsere späteren Betrachtungen voraussetzen werden.

Ein *Cluster–Graph* oder *Clustergraph* oder *geclusterter Graph* $C = (G, T)$ besteht aus einem ungerichteten Graphen $G = (V_G, E_G)$ und einem Baum $T = (V_T, E_T)$ mit einer Wurzel, so daß die Blätter von T genau die Knoten des Graphen G sind. Ein *Cluster* ist eine Knotenteilmenge im Graphen G.

Für die Knotenmenge des Graphen G und die Knotenmenge des Baumes T gilt somit

$$V_G \subset V_T. \tag{1}$$

Sei $\nu \in T$ ein Baumknoten, aber kein Blattknoten. Sei die Knotenteilmenge $V_G^{Cluster}$ mit

$$V_G^{Cluster} \subset V_G \tag{2}$$

ein Cluster des Graphen G. Sei der Baum $T(\nu)$ ein vom Baumknoten ν induzierter Baum. Sei lca der kleinste gemeinsame Vorfahre zweier Baumknoten in T. Ferner soll gelten

$$\nu = lca(v, w) \text{ für alle } v, w \in V_G^{Cluster} \text{ paarweise verschieden.}$$

Dann wird das Cluster $V_G^{Cluster} = V_G(\nu)$ im Baum T durch den Baumknoten ν repräsentiert. In dieser Arbeit ist mit einem Cluster ν aus dem Baum T immer die zu dem Cluster korrespondierende Wurzel $\nu \in T(\nu) \subset T$ gemeint.

Eine Kante ist *inzident* zu einem Cluster ν, wenn die Kante eines Knotens aus $V(\nu)$ einen Knoten außerhalb von $V(\nu)$ verbindet. Eine Kante heißt *Auswärtskante* (bzw. *Inwärtskante*) eines Clusters ν, falls sie Auswärtskante (bzw. Inwärtskante) eines Knotens aus $V(\nu)$, aber nicht Inwärtskante (bzw. Auswärtskante) eines Knotens aus $V(\nu)$ ist.

Für jeden Knoten $\nu \in T$ gilt: $chl(\nu)$ sind die *Kinder* von ν; $pa(\nu)$ der *Vater* von ν , falls ν nicht die Wurzel von T ist. Jeder Knoten ν des Baumes T repräsentiert einen *Cluster* $V(\nu)$

der Knoten von G, die die Blätter des Unterbaumes mit Wurzel ν sind. Der *Untergraph von* G, der durch $V(\nu)$ induziert wird, wird als $G(\nu)$ bezeichnet.

Damit beschreibt der Baum T eine Inklusionsrelation zwischen den Clustern.

Wenn ein Knoten ν' Nachkomme eines Knotens ν im Baum T ist, so bezeichnet man das Cluster von ν' als *Subcluster* von ν.

In einer Zeichnung eines Cluster–Graphen $C = (G, T)$ werden die Knoten des Graphen G wie üblich als Punkte und die Kanten als Kurven in einer Ebene gezeichnet.

Ferner soll für eine Zeichnung des Graphen in einer Ebene gelten, daß für alle ν aus T das Cluster als ein einfach geschlossenes Gebiet R gezeichnet werden soll, welches im Innern die Zeichnung von $G(\nu)$ enthält, so daß:

1. die Gebiete aller Untercluster von ν ganz im Innern von R enthalten sind;
2. die Gebiete aller anderen Cluster ganz im Äußeren von R enthalten sind;
3. für jede Kante e zwischen zwei Knoten von $V(\nu)$ die Zeichnung von e ganz in R enthalten ist.

Diese Bedingungen erleichtern die Verständlichkeit einer Zeichnung. Hieraus wird nun der Begriff der c–Planarität abgeleitet.

Wir gehen im folgenden davon aus, dass die Cluster–Regionen als ein geschlossenes Gebiet dargestellt werden.

Schneidet eine Kante e die Umrandung eines Gebiets R in der Zeichnung von (G, T) mehr als einmal, dann haben die Kante e und das Gebiet R in dieser Zeichnung einen *Kanten–Gebiet–Schnitt*.

Eine Zeichnung eines Cluster–Graphen ist *c–planar (compound planar)*, falls es keine Kantenkreuzungen oder Kanten–Gebiet–Schnitte gibt.

Ein Cluster–Graph C ist *c–planar* genau dann, wenn er eine c–planare Zeichnung hat.

Eine *c–planare Einbettung* von $C = (G, T)$ besteht aus einer planaren Einbettung von G sowie für jedes nichttriviale Cluster (das mehr als einen Knoten enthält) der kreisförmigen Ordnung in welcher die Inzidenzkanten die Umrahmung des Clustergebiets kreuzen.

Die Planarität eines zugrundeliegenden Graphen G impliziert nicht die Existenz einer c-planaren Zeichnung D eines Cluster-Graphen C.

D.h. ein c-planarer Cluster-Graph ist ein planarer Cluster-Graph, aber nicht jeder planare Cluster-Graph ist ein c-planarer Cluster-Graph. c-Planare Cluster-Graphen bilden somit eine Teilklasse der planaren Cluster-Graphen.

In einem Cluster-Graphen kann es vorkommen, daß der durch ein Cluster induzierte Untergraph nicht zusammenhängend ist, sogar wenn der zugrundeliegende Graph zusammenhängend ist. Dieses macht die Betrachtung schwieriger, als es bei klassisch planaren

Graphen der Fall ist, da es nunmehr mehrere Möglichkeiten für die Plazierung und Formung der Regionen gibt, sogar bei gegebener fixer Einbettung.

Sei nun C ein Cluster-Graph, in dem der von jedem Cluster induzierte Untergraph zusammenhängend ist. Wir können o.B.d.A. annehmen, daß in T mit Ausnahme der Blätter jeder Knoten mindestens zwei Kinder hat.

Ein Cluster-Graph $C = (G, T)$ ist ein *c-zusammenhängender Cluster-Graph*, falls jedes Cluster einen zusammenhängenden Untergraphen von G induziert.

Das folgende Theorem 1 von Feng, Eades, Cohen [30] gibt uns ein notwendiges wie auch ein hinreichendes Kriterium für die c-Planarität eines c-zusammenhängenden Cluster-Graphen.

Theorem 1 (Feng, Eades, Cohen [30]). *Ein c-zusammenhängender Cluster-Graph ist c-*planar *genau dann, wenn der Graph G planar ist und wenn eine planare Zeichnung D von G existiert, so daß für jeden Knoten ν von T alle Knoten und Kanten aus $G \backslash G(\nu)$ in der äußeren Region der Zeichnung von $G(\nu)$ liegen.*

Auf dieses Theorem 1 stützt sich der erste Algorithmus des Planaritätstests für c-zusammenhängende Cluster-Graphen von Feng, Eades, Cohen [30] mit quadratischer Laufzeit in bezug auf die Anzahl der Eingabeknoten mit der Voraussetzung, dass jedes Cluster mindestens zwei Kinder hat. Verbessert wurde dieser auf lineare Laufzeit von Elias Dahlhaus [18, 20].

Unser Augenmerk richtet sich auf das Theorem 2 von Feng, Eades, Cohen [30] welches die Voraussetzung für unsere Betrachtungen ist. Dieses Theorem gibt uns eine Charakterisierung der c-Planarität für allgemeine Cluster-Graphen. Dazu benötigen wir zunächst eine weitere Definition (Feng, Eades, Cohen [30]):

Seien $C_1 = (G_1, T_1)$ und $C_2 = (G_2, T_2)$ zwei Cluster-Graphen in dem Sinne, daß T_1 ein Unterbaum von T_2 ist und $G_1(\nu)$ ein Untergraph von $G_2(\nu)$ ist. Dann ist C_1 ein *geclusterter Untergraph* von C_2 und C_2 ein *geclusterter Supergraph* von C_1.

Theorem 2 (Feng, Eades, Cohen [30]). *Ein Cluster-Graph $C = (G, T)$ ist c-planar genau dann, wenn er ein geclusterter Untergraph eines c-zusammenhängenden und c-planaren Cluster-Graphen ist.*

Es ist bislang kein effizienter (polynomieller) Algorithmus bekannt, der nicht c-zusammenhängende Cluster-Graphen zu c-zusammenhängende Cluster-Graphen erweitert. Überdies ist kein effizienter Algorithmus bekannt, der nicht c-zusammenhängende Cluster-Graphen auf c-Planarität testet.

Um die Komplexität dieser Probleme zu klären, folgen weitere Arbeiten, die in Kooperation mit Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel und René Weiskircher entstanden [41, 40]; siehe Kapitel 6 und Kapitelabschnitte 7.3 bis 7.5 dieser Dissertation. Da c-zusammenhängende Cluster–Graphen auf c–Planarität effizient in polynomieller Zeit getestet werden können, ist es intuitiv ersichtlich, zuerst einen c-Zusammenhang zu generieren, ohne eine vorhandene c-Planarität zu zerstören. Da dieses bekanntermassen in bezug auf Komplexität unbekannt ist, gehen wir zuerst nur von einem nicht zusammenhängenden Cluster aus. Ein dazu verwandtes Problem ist das folgende: Gegeben sei ein planarer Graph und eine echte Teilmenge W aus Knoten enthalten im Graphen. Der durch die Knoten aus W induzierte Untergraph G_W sei nicht zusammenhängend. Addiere minimale Anzahl Kanten hinzu, so dass, falls möglich, G_W zusammenhängend wird und G planar bleibt. Das besondere hierbei ist, dass wir über alle Einbettungen betrachten, die in deren Anzahl exponentiell viele sein können. Erstaunlich ist, dass dieses in linearer Zeit realisiert werden kann. Dieses Problem ist als das SIPCA-Problem bekannt [41]; siehe Kapitel 6 dieser Arbeit. Minimale Anzahl Kanten wird verlangt, da wir für die Addition in einen c–planaren Cluster–Graphen keine Kanten hinzufügen möchten, die im induzierten Untergraphen eines Clusters einen Kreis schliessen und so eine vorhandene c–Planarität verletzt werden könnte. Mit Hilfe von SIPCA lassen sich eine größere Klasse Cluster–Graphen testen [40]; siehe Kapitel 7.3 bis 7.5 dieser Arbeit. Es verbleiben solche mit den folgenden Eigenschaften: es existieren zwei nicht zusammenhängende Geschwistercluster, wobei die Möglichkeiten der Kantenhinzufügung des einen Clusters durch mindestens eine Kantenhinzufügung des anderen eingeschränkt wird.

Für solche wird die SIPCA-Einsetzung problematisch. Frage hierbei ist, welche Kantenhinzufügung bei dem einen Cluster erlaubt sind, so dass die Verbindungsmöglichkeiten des anderen Clusters nicht allzu eingeschränkt werden. Hier werden graphentheoretische Ansätze in dieser Arbeit analysiert. Resultierend aus den Betrachtungen ergibt sich ein polynimieller Algorithmus für planare Cluster-Graphen für die $G - G(\nu)$ für jedes Cluster ν zusammenhängend ist; siehe Kapitel 7.2.

Offen bleibt das Problem u. a. für seriell-parallele Cluster-Graphen, bei denen die Knoten in bezug auf ihre Clusterzugehörigkeit nicht auf Schichten platziert werden können, sondern kreuz und quer im Graphen liegen. Hier existieren exponentiell viele Möglichkeiten der Kantenhinzufügung und planarer Einbettung. Offen bleibt es auch für Spezialfälle, wobei der Graph nicht zusammenhängend ist und über die Zusammenhangskomponenten nicht zusammenhängende Cluster verlaufen. Hier können genauso exponentiell viele Möglichkeiten existieren, die Zusammenhangskomponenten so auszurichten, dass bei einem c-planaren Cluster–Graphen auch eine c–planare Einbettung erstellt wird. Hier erkennen wir, dass die Einschränkung des Planaritätsbegriffs auch Einschränkungen in Augmentationen der Cluster-Graphen mit sich zieht. In dieser Hinsicht wurden auch weitere Kantenerweiterungen für Cluster–Graphen studiert. Spannend ist auch die Frage nach Cluster–Graphen, die c–planar sind, sobald sie planar sind. Aus der Literatur bekannt sind bisher c–zusammenhängende Cluster–Graphen deren Graph ein Baum ist [21], bipartite Graphen [30], und c–zusammenhängende Cluster–Graphen, für die $G - G(\nu)$ für jedes Cluster auch zusammenhängend ist; siehe [15, 54] und Kapitel 8 dieser Dissertation. Auch studiert werden verbotene Untergraphen eines c-zusammenhängenden c–planaren Cluster–Graphen, natürlich neben $K_{3,3}$ und K_5 und somit eine neue Charakterisierung für solche Cluster-Graphen gefunden; siehe Kapitel 7.1.

In Kapitel 8 wird eine Methode vorgestellt, die c-planar eingebettete Cluster-Graphen so mit Kanten erweitert, so daß jede allgemeine Kantenaddition von Graphen auf den Cluster-Graphen angewendet werden kann ohne seine c-planare Einbettung zu zerstören.

Im zweiten Teil der Dissertation beschäftigen wir uns mit dem Maximum Planaren Untergraphenproblem, das sich formulieren läßt als die Suche nach dem größten - im Sinne der Kantenanzahl - Untergraphen eines gegebenen Graphen. In Kapitelabschnitt 9.1 zeigen wir, daß das Maximum Planare Untergraphenproblem $\mathcal{NP}$-schwer bleibt auf nicht-planaren Graphen ohne Minoren isomorph entweder zu $K_{3,3}$ oder K_5. Das Studium möglicher polynomieller Spezialfälle unter (Tutte) Dekomposition [81] ergab ein neues Problem zur Reduzierung des Zusammenhangs in bezug auf 3 Knoten (resultierend aus der Betrachtung von Δ-Summen [81]). Dieses Zusammenhangsreduktionsproblem wird in Kapitelabschnitt 9.2 diskutiert, und wird als das gewurzelte $K_{1,3}$- bzw. $K_{2,3}$-Minorenlöschungsproblem bezeichnet. In Kapitel 10 schliessen wir den zweiten Teil der Dissertation mit der Betrachtung von der Kreuzungsreduzierung in gerichteten Graphen unter der Nebenbedingung daß in der resultierenden Planarisierung für jeden Knoten alle eingehenden Kanten von den ausgehenden getrennt sind, die sogenannte bimodale Kreuzungsminimierung. Wir können experimentell zeigen, daß für praktische Instanzen (ROM-Graphen [74]) die Anzahl der Kreuzungen in dieser speziellen Planarisierung im Vergleich zur allgemeinen Planarisierung nicht signifikant wächst.

Im dritten Teil der Dissertation geben wir eine Diskussion und Ausblick auf zukünftige geplante Forschungsarbeiten des ersten Teils der Dissertation.

Contents

Chapter 1

Introduction

Visualization of structural information is getting increasing attention. Graphs are widely used to model relational structure such as networks.

A graph drawing algorithm takes as input a graph and computes a layout of the graph in two or three dimensional space by assigning each vertex to a point and mapping each edge to a simple curve.

The visualization of graphs is getting more and more complex. Consequently, extended structured graphs variation is needed. That is why various graphs extensions are developed such as hypergraphs, compound graphs, higraphs and cigraphs. Hypergraphs define relations over sets of entitles that have edges in compassing several vertices. Higraphs are derived from hypergraphs with “blobs” or “sub-blobs” that can intersect or enclose one another. Compound graphs allow for inclusion relations and adjacency relations but are weaker than higraphs. In total, it seems to be some difficulty in finding drawing algorithms for these graph types. Only heuristic methods has been established for the hierarchical layout of compound digraphs.

A subclass of compound graphs are the so-called clustered graphs. A clustered graph consists of a graph G and a recursive partitioning of the vertices of G. Each partition is a cluster of a subset of the vertices of G. Clustered graphs are getting increasing attention in graph drawing [18, 21, 20, 25, 30]. Formally, a *clustered graph* $C = (G, T)$ is defined as an undirected graph G and a rooted tree T in which the leaves of T correspond to the vertices of $G = (V, E)$.

In a cluster drawing of a clustered graph, vertices and edges are drawn as usual, and clusters are drawn as simple closed curves defining closed regions of the plane. The region of each cluster C contains the vertices W corresponding to C and the edges of the graph induced by W. The borders of the regions for the clusters are pairwise disjoint. If a cluster drawing does not contain crossings between edge pairs or edge/region pairs, we call it a *c-planar* drawing. Graphs that admit such a drawing are called *c-planar*.

Drawing clustered graphs is required in many applications. For example in information systems: in process analysis, processes are clustered into other processes at different abstraction levels. An other example are social networks, the actors are typically grouped into classes of actors with affine features.

While the complexity status of c-planarity testing is unknown, the problem can be solved in linear time if the graph is c-connected, i.e., all cluster induced subgraphs are connected [18, 20, 30]. In approaching the general case, it appears natural to augment the clustered graph by additional edges in order to achieve c-connectivity without loosing c-planarity.

Somehow, this seems to be still difficult when we take a look at Figure 1.1: The question that

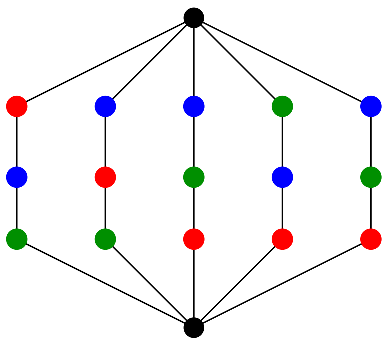


Figure 1.1: A planar clustered graph: Is it c-planar?

occurs here is whether the presented clustered graph is c-planar. Somehow, this appears hard to say directly even if the clustered graph is very small. If we take a closer look, we see that neither the subgraphs induced by the clusters nor the subgraphs induced by $G - G(\nu)$ for each cluster ν are connected. This fact seems to open a huge number of possibilities for the edge augmentation described above. After trying all possibilities we recognize that the clustered graph of Figure 1.1 is not c-planar. The complexity status of this case is still open - but we show in this thesis that the case where for each cluster ν $G - G(\nu)$ is connected can be solved in polynomial time beside some other special cases.

That is why we do a step more. In Part I of this thesis, we focus on augmenting the clustered graph such that it remains c-planar, get c-connected *and* the subgraph induced by $G - G(\nu)$ get connected for each cluster ν. The clustered graph is c-planar if and only if it remains planar.

Most of the algorithms in Part I of this thesis use ideas from the linear time algorithm for subgraph induced planar connectivity augmentation presented in Chapter 6 [41]. For an undirected graph $G = (V, E)$, $W \subseteq V$, and $E_W = \{(v_1, v_2) \in E : \{v_1, v_2\} \subseteq W\}$ let $G_W = (W, E_W)$ be the subgraph of G induced by W. If G is planar, a *subgraph induced planar connectivity augmentation* for W is a set F of additional edges with end vertices in W such that the graph $G' = (V, E \cup F)$ is planar and the graph G'_W is connected.

While Part I of this thesis focus on planarity and augmentation of clustered graphs, Part II of this thesis deals with edge deletion and crossing minimization on graphs. Part III gives some future work ideas on Part I.

Part I-III are organized as follows: After an introduction in the technical foundations in Chapter 2, we start with Part I on Planarity and Planar Augmentation. First, we give a short overview on the state of the art in the field of clustered graphs. In Chapter 4 we consider the computational complexity of deciding c-planarity. Then we give some characterizations on c-planarity in Chapter 5. In Chapter 6 we discuss the subgraph induced planar connectivity augmentation that we use in the following Chapter 7 to derive polynomial running time algorithms for some subclasses of planar clustered graphs. Then, in Chapter 8 we discuss an augmentation method for c-planar embedded clustered graphs.

In Part II we deal with minimum edge deletion and bimodal crossing minimization. In Chapter 9 we show that the maximum planar subgraph problem remains $\mathcal{NP}$-hard even for $K_{3,3}$- or K_5-minor free non-planar graphs. Furthermore, we consider the rooted $K_{i,3}$-minor minimum deletion problem for $i = 1, 2$. The work of this chapter is motivated by exploring the maximum planar subgraph problem under (Tutte) connectivity decomposition [81] and the results are the first steps. In Chapter 10 we do some experimental considerations for the bimodal crossing minimization that turn out that for practical graphs the number of crossings is not increased significantly.

Finally, in Part III we conclude, summarize remaining open problems and give some ideas for future work concerning Part I.

Chapter 2

Preliminaries

2.1 Graphs and Their Representation

In this section we use mostly the definitions presented by Jünger, Mutzel in [56].

2.1.1 Undirected Graphs

A *graph* $G = (V, E)$ consists of a finite set $V = V(G)$ of *vertices* or *nodes* and a finite set $E = E(G)$ of *edges*. An edge $e = (v, v)$ is called a *loop* and if for two edges $e_1, e_2 \in E$ we have $e_1 = (v, w) = e_2$ we say that e_1 and e_2 are multi-edges. Figure 2.1 shows a graph with a loop and a pair of multi-edges.

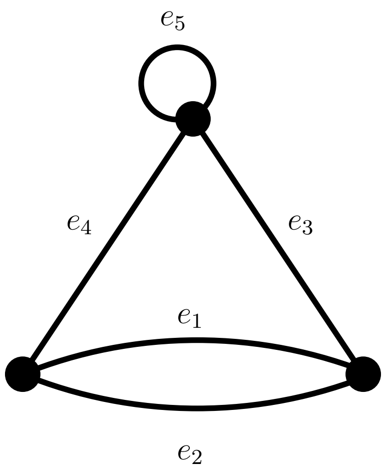


Figure 2.1: A graph with a loop e_1 and two multi-edges e_2, e_3 [56].

A graph with no loops and no multi-edges is characterized by a finite set V of vertices and a finite set $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$ of edges and called a *simple graph*.

For an edge $e = \{u, v\}$, the vertices u and v are the *end-vertices* of e, and e is *incident* to u and v. An edge $\{u, v\} \in E$ *connects* the vertices u and v. Two vertices $u, v \in V$ are *adjacent* if $\{u, v\} \in E$. By $\text{star}(v) = \{e \in E \mid v \in e\}$ we denote the set of edges incident to a vertex $v \in V$ and $\text{adj}(v) = \{u \in V \mid \{u, v\} \in E\}$ is the set of vertices adjacent to a vertex $v \in V$. By $\deg(v) = |\,\text{star}(v)| + |\,\text{loop}(v)|$, where $\text{loop}(v)$ is the set of edges of the form $\{v, v\}$, we denote the *degree* of a vertex $v \in V$, $\text{mindeg}(G) = \min\{\deg(v) \mid v \in V\}$ is the *minimum degree* and $\text{maxdeg}(G) = \max\{\deg(v) \mid v \in V\}$ is the *maximum degree* of G. E.g., in Figure 2.1, $\text{star}(v_1) = \{e_1, e_2, e_3\}$, $\text{star}(v_3) = \{e_3, e_4, e_5\}$, whereas $\text{adj}(v_1) = \{v_2, v_3\}$ and $\text{adj}(v_3) = \{v_1, v_2, v_3\}$. The degrees of these two vertices are $\deg(v_1) = 3$ and $\deg(v_3) = 4$, the minimal degree of the graph is $\text{mindeg}(G) = 3$ and the maximum degree is $\text{maxdeg}(G) = 4$. A vertex v with $\deg(v) = 0$ is called an *isolated vertex*.

For $W \subseteq V$ let $E[W] = \{\{u, v\} \in E \mid u, v \in W\}$ and for $F \subseteq E$ let $V[F] = \{v \in V \mid v \in e \text{ for some } e \in F\}$. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ or *contained* in G if $V' \subseteq V$ and $E' \subseteq E$. For a vertex set $W \subseteq V$ we call $G[W] = (W, E[W])$ a *vertex-induced subgraph* of G and for an edge set $F \subseteq E$ we call $G[F] = (V[F], F)$ an *edge-induced subgraph* of G.

A *path* W of *length* k in a graph G is an alternating sequence of distinct vertices and edges $v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k$, *beginning* and *ending* with the vertices v_0 and v_k, respectively, and $e_i = \{v_{i-1}, v_i\}$ for $i = 1, 2, \ldots, k$. A path is called a *cycle* if all vertices are distinct except for $v_0 = v_k$ and $k \geq 2$. A graph that does not have any cycles is called a *forest*.

A *coloop* is an edge in a graph that does not lie in any cycle.

A graph G is *connected* if every pair of vertices is connected by a path, otherwise it is called *disconnected*. A *component* of G is a maximal connected subgraph of G. Consequently, a disconnected graph has at least two components. A connected forest G is called a *tree*.

A graph $G = (V, E)$ is *k-connected* if at least k vertices must be removed from V in order to make the resulting vertex-induced subgraph disconnected. By $\kappa(G) = \max\{k \mid G \text{ is } k\text{-connected}\}$ we denote the *(vertex-)connectivity* of G.

The following definitions are given by Truemper [81]: Let $G = (V, E)$ be a connected graph. Let (E_1, E_2) be a pair of nonempty sets that partition the edge set E. Let G_1 (resp. G_2) be obtained by removal of the edges E_2 (resp. E_1). We assume G_1 and G_2 to be connected. We suppose that pairwise identification of k vertices of G_1 with k vertices of G_2 produces G. (E_1, E_2) is a (Tutte) k-separation if E_1 and E_2 have at least k edges each. G is called (Tutte) 3-connected if it has no (Tutte) 1- or 2-separation. G is called a *2-sum* (composition) of the connected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, denoted $G = G_1 \oplus_2 G_2$, if the following process yields G: we identify an arbitrary edge e_1 of G_1 with an arbitrary edge e_2 of G_2 and delete this edge in $G_1 \cup G_2$.

Of special interest in automatic graph drawing are 1, 2, and 3-connected graphs, also called *connected*, *biconnected*, and *triconnected* graphs, respectively. A vertex whose removal disconnects the graph is called a *cut vertex*, i.e., a graph is biconnected if it has no cut vertex. The maximal biconnected components of a graph G are called the *blocks* of G. The blocks intersect in cut vertices, see Figure 2.2 for an illustration.

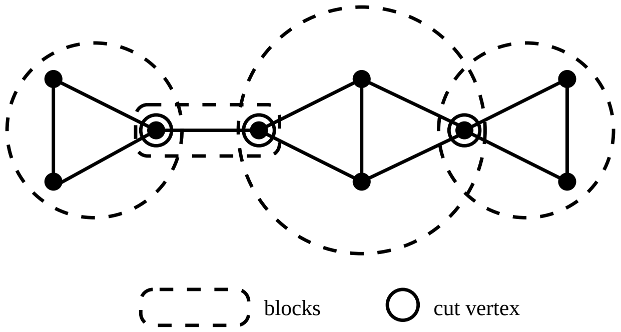


Figure 2.2: The cut vertices and the blocks of a graph [56].

Two vertices whose removal disconnects a biconnected graph are called a *separating vertex pair*, i.e., a graph is triconnected if it has no separating vertex pair.

An edge whose removal disconnects the graph is called a *bridge*. The graph in Figure 2.2 contains exactly one bridge.

A graph $G = (V, E)$ is *k-edge-connected* if at least k edges must be removed from E in order to make the resulting edge-induced subgraph disconnected. By $\lambda(G) = \max\{k \mid G \text{ is } k\text{-edge-connected}\}$ we denote the *edge-connectivity* of G.

A *series-parallel graph* is defined recursively [81]. Starting with a single edge $e = (v, w)$, we can in each step either replace e by a path with two edges with end vertices v and w (that means by splitting e with a vertex) or with two multiple edges. Doing this recursively, all possible graphs that can be constructed by this approach is a series-parallel graph.

It is obvious that series-parallel graphs are planar.

An *outer-planar graph* is a graph without a minor isomorphic to K_4 and $K_{1,3}$ that is a series-parallel graph without a minor isomorphic to $K_{1,3}$ [81].

2.1.2 Directed Graphs

A *directed graph* or *digraph* $G = (V, E)$ consists of a finite set $V = V(G)$ of *vertices* and a finite multi-set $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ of (*directed*) *edges* or *arcs* that are ordered pairs of vertices. Omitting for every edge the order of its vertices, we get an undirected graph. Clearly, concepts like, e.g., subgraph, path, cycle, forest, component, or tree, inherit to directed graphs.

An *acyclic digraph (directed acyclic graph: dag)* is a digraph with no directed cycle or loop. If $e = (u, v)$ then e is an *outgoing* or *leaving* edge of u and an *incoming* or *entering* edge of v. By $\text{instar}(v) = \{(u, v) \in E \mid u \in V\}$ we denote the set of incoming edges of a vertex $v \in V$ and by $\text{outstar}(v) = \{(v, u) \in E \mid u \in V\}$ we denote the set of outgoing edges of a vertex $v \in V$. Accordingly, we define $\text{inadj}(v) = \{u \in V \mid (u, v) \in E\}$ and $\text{outadj}(v) = \{u \in V \mid (v, u) \in E\}$. Then $\text{indeg}(v) = |\,\text{instar}(v)|$ is the *in-degree* and $\text{outdeg}(v) = |\,\text{outstar}(v)|$ is the *out-degree* of a vertex $v \in V$.

A *source* is a vertex with no incoming edges and a *sink* is a vertex with no outgoing edges. An acyclic digraph G with exactly one source is called a *single source directed graph* digraph. If, in addition, its corresponding undirected graph is connected and has no loop and no (undirected) cycle, the graph is called a *rooted tree* whose *root* is the only vertex $v = \text{root}(T) \in V$ with $\text{indeg}(v) = 0$ and whose *leaves* are vertices $v \in V$ with $\text{outdeg}(v) = 0$. The *depth* $\text{depth}(v)$ of a vertex v in a rooted tree $T = (V, E)$ is the length of the (unique) directed path from the root of T to v. All vertices of depth k constitute *tree level* k. Furthermore, for each $v \in V$ that is not a leaf, the vertices in $\text{outadj}(v)$ are called *children* of v, and for each $v \in V$ other than the root, the vertex in $\text{inadj}(v)$ is called *parent* of v. Children of the same parent are called *siblings*. An acyclic digraph with exactly one sink is called a *single sink* digraph. An acyclic digraph with exactly one source s and exactly one sink t and an edge (s, t) is called an *st-digraph*.

A *series-parallel digraph* is a directed graph with the following recursive definition:

- A graph consisting of a source s, and a sink t, and a directed edge from s to t is a series-parallel digraph.

- If G' and G'' are series-parallel digraphs, the series composition obtained by identifying the sink of G' with the source of G'' is also a series-parallel digraph.

- If G' and G'' are series-parallel digraphs, the parallel composition obtained by identifying the source of G' with the source of G'' and by identifying the sink of G' with the sink of G'' is also a series-parallel digraph.

A *topological numbering* of G is an assignment of numbers $\text{topnumber}(v)$ to the vertices v of G such that for every edge (u, v) of G the number assigned to v is greater than the one assigned to u (i.e., $\text{topnumber}(v) > \text{topnumber}(u)$). A *topological sorting* of G is a topological numbering of G such that every vertex is assigned a distinct integer between 1 and $|V|$. It is easy to see that G admits a topological numbering or sorting if and only if G is acyclic.

2.1.3 Representation of Graphs

There are several ways to represent an (undirected or directed) graph. In this thesis, we restrict our attention to representation in 2-dimensional space. A graph $G = (V, E)$ is generally visualized by a *drawing* in 2 or 3-dimensional space with the vertices drawn as points or boxes of a pre-specified width and height, and the edges drawn as closed Jordan curves, connecting their incident vertices. Layouts in which the coordinates of the vertex representations are restricted to integer values are called *grid layouts*.

2.1.4 Clustered Graphs

Clustered graphs are graphs with recursive clustering structures over the vertices [30]. A *clustered graph* $C = (G, T)$ consists of an undirected graph $G = (V, E)$ and a rooted tree T, the *cluster-tree* or *clustertree*, such that the leaves of T are exactly the vertices of G. Each vertex ν of T represents a *cluster* $V(\nu)$ of the vertices of G that are the leaves of the subtree rooted at ν. The root of T is called *root cluster*. The tree T is called the *inclusion tree* of C because it describes an inclusion relation between clusters. The graph G is called the *underlying graph* of C. The tree $T(\nu)$ represents the subtree of T rooted at the vertex ν, and $G(\nu)$ denotes the subgraph of G induced by the cluster associated with vertex ν. We define $C(\nu) = (G(\nu), T(\nu))$ to be the *sub-clustered graph* associated with vertex ν and C to be the super-clustered graph of $C(\nu)$. For example, suppose that $C_1 = (G_1, T_1)$ and $C_2 = (G_2, T_2)$ are two clustered graphs such that T_1 is a subtree of T_2, and for each node ν of T_1, $G_1(\nu)$ is a subgraph of $G_2(\nu)$. Then C_1 is a sub-clustered graph of C_2, and C_2 is a super-clustered graph of C_1. An edge $\{v, w\} \in E$ with $v \in V(G(\nu))$ and $w \in V \setminus V(G(\nu))$ is said to be *incident* to cluster ν.

A *cut cluster of* C is a non-trivial cluster ν of C that becomes a cut vertex by shrinking. It is an *embedded isolating cut cluster* in a given planar embedding of C if it induces a cycle in $G(\nu)$ that isolates $G - G(\nu)$ in the embedding. It is called an *isolating cut cluster* if it is an embedded isolating cut cluster for all planar embeddings of C.

A clustered graph $C = (G, T)$ is *connected* if G is connected. A clustered graph $C = (G, T)$ is *c-connected* if each cluster induces a connected subgraph of G. A clustered graph is *cluster-biconnected* if and only if for every (trivial and non-trivial) cluster ν $G - G(\nu)$ is connected. Observe that G is then biconnected. We formulate a weaker version of cluster-biconnectivity in the case that only for non-trivial clusters ν $G - G(\nu)$ is connected. For simplicity we call the clustered graphs that satisfy the weaker version *co-connected* in this thesis. A clustered graph that is c-connected and co-connected is called *completely connected* clustered graph [15].

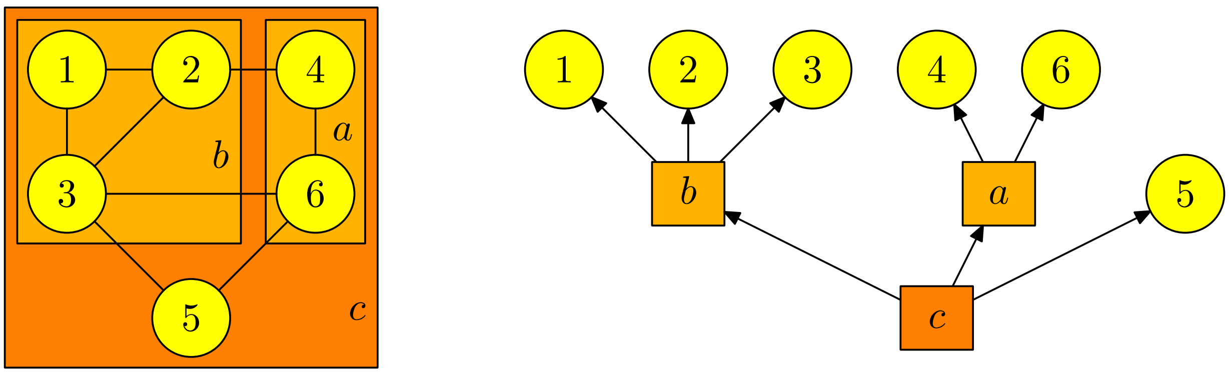


Figure 2.3: A drawing of a clustered graph and its defining tree [56].

In a *drawing of a clustered graph* $C = (G, T)$, the graph G is drawn with points and curves as usual. For each vertex ν of T, the cluster is drawn as a simple closed region R (i.e., a region without holes) that contains the drawing of $V(G(\nu))$, such that the following three conditions hold.

(i) The regions for all sub-clusters of ν are completely contained in the interior of R.

(ii) The regions for all other clusters are completely contained in the exterior of R.

(iii) If there is an edge e between two vertices of $V(\nu)$ then the drawing of e is completely contained in R.

Figure 2.3 shows a drawing of a clustered graph $C = (G, T)$ and the corresponding tree T.

We say that there is an *edge-region crossing* in the drawing if the drawing of edge e crosses the drawing of region R more than once.

2.1.5 Compound Graphs

Compound graphs have been introduced for representing graphs with both inclusion and adjacency relationships [78]. A *compound graph* $C = (G, T)$ is defined as an (undirected or directed) graph $G = (V, E_G)$ and a rooted tree $T = (V, E_T)$ that share the same vertex set V. There is a one to one correspondence between the structure of the tree and the set of inclusions between the vertices, namely, a vertex u is in direct inclusion relation to v if and only if u is a child of v in the tree. If the end vertices u and v of all edges $\{u, v\} \in E_G$ belong to different root-leaf paths in T, C is called a *simple compound graph* . In a simple compound graph, a pair of vertices (u, v) cannot be in an adjacency and in an inclusion relation at the same time. Figure 2.4 shows an example of a simple compound graph.

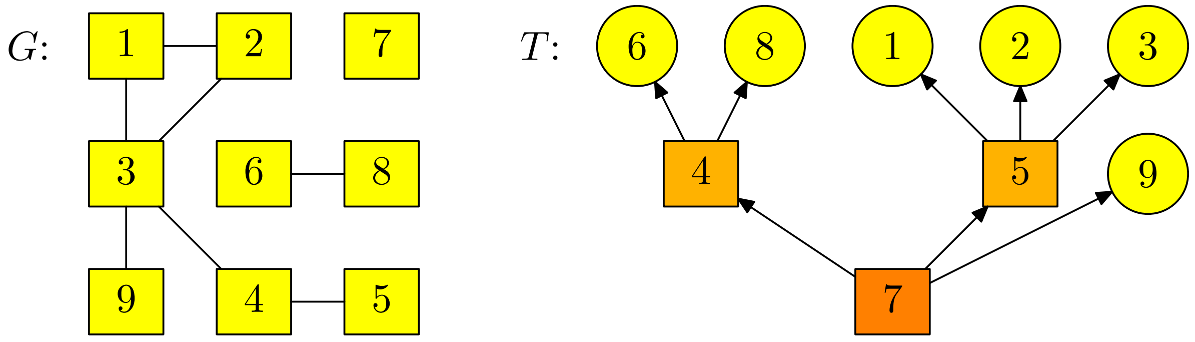


Figure 2.4: A compound graph defined by a graph and a tree [56].

Edges connecting vertices of different tree levels are called *inter-level edges*. If a compound graph does not contain inter-level edges, we call it a *nested graph*.

A further restriction allowing only edges between the leaves of the tree leads to an alternative definition of clustered graphs (see Section 2.1.4).

In a *drawing of a compound graph* $C = (G, T)$, the vertices of the graph G are drawn as closed regions so that a vertex u is included in the region representing the vertex parent(u) in T, and the edges in E_G are drawn as curves connecting the regions associated with its end-vertices. Figure 2.5 shows a drawing of the compound graph defined in Figure 2.4.

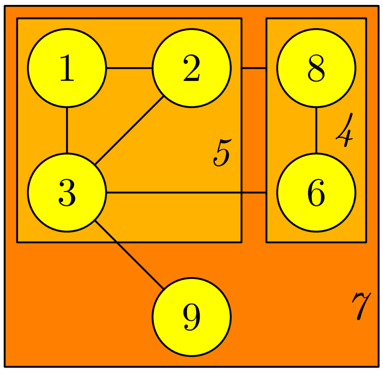


Figure 2.5: A drawing of the compound graph defined in Figure 2.4 [56].

2.2 Graph Planarity and Embeddings

This section deals with drawings and embeddings of a graph onto the plane. A drawing of a graph G on the plane yields an *embedding* Π of G, i.e., a clockwise ordering of the incident edges for every vertex with respect to the drawing.

2.2.1 Planar Graphs

A graph $G = (V, E)$ is called *planar* if it can be drawn in the plane such that no two edges cross each other except at common endpoints. An intersection of two edges in a drawing other than at their endpoints is called a *crossing*. A *planar* or *combinatorial embedding* Π of a planar graph G is an embedding with respect to a planar drawing. A graph with a given fixed planar embedding Π is also called a *plane graph*. Given a drawing of a plane graph G, a *face* of G is a topologically connected region in the drawing bounded by the (Jordan curves corresponding to the) edges of G. A face of a plane graph is uniquely described by its boundary edges. The degree deg(f) of a face f is defined as the number of its boundary edges, where each boundary edge with both sides on the boundary of f is counted twice. The faces of a plane graph are already described by the planar embedding. Two faces are *adjacent* if their boundaries share an edge. The one unbounded face of a plane graph is called the *outer face* or *exterior face*. All other faces are called *interior faces*. An equivalent definition of a planar embedding is an ordered list of the boundary edges for each face, clockwise for interior faces and counter-clockwise for the exterior face.

A famous result of Euler [27] for polytopes relates the number of vertices, edges, and faces in any planar embedding of a connected planar graph:

Theorem 2.1 (Euler's Formula [27])**.** *Let Π be a planar embedding of a connected planar graph $G = (V, E)$ and let F be the set of faces in Π. Then $|V| - |E| + |F| = 2$.*

From Euler's formula, an upper bound on the number of edges of a planar graph with a given number of vertices is easily derived:

Theorem 2.2. *For any simple planar graph $G = (V, E)$ with at least 3 vertices we have $|E| \leq 3|V| - 6$.*

The bound is attained for *triangulated* planar graphs, i.e., planar graphs in which every face is a triangle.

While, in general, the number of different planar embeddings of a planar graph is exponential in $|V|$, a triconnected planar graph has only two different planar embeddings, which are mirror-images of each other.

A minor can be defined in the following way: Let G and H be two undirected graphs. H is a minor of G if there exists a subgraph H' of G and a partition $V(H') = V_1 \uplus \cdots \uplus V_k$ of its vertex set into connected subsets such that contracting each of $V_1, \ldots, V_k$ yields a graph isomorphic to H.

Wagner characterizes a planar graph as a graph that has no K_5 and $K_{3,3}$ minors [84]. His theorem is a significant reformulation of Kuratowski's well-known result [58].

Whitney's 2-isomorphism theorem [81] says that two graphs have *isomorphic cycle spaces* if and only if one can be transformed into the other by *Whitney flips.* There are two types of these flips:

- 1-*flips:* either identify two vertices that lie in different components of G or which are cut vertices embeddable on the same face,
- 2-*flips:* decompose a graph G along two vertices v, w into two subgraphs G_v and G_w and then compose them again by identifying v in G_v with w in G_w and vice versa.

Observe that adding or deleting a vertex with degree 0 is a 1-flip.

If two graphs have isomorphic cycle spaces we say that they are 2-*isomorphic*.

Given a planar embedding $\Pi(G)$ of a planar graph $G = (V, E)$ with face set F, the *dual graph* $G' = (V', E')$ is constructed as follows:

The nodes of G' are the faces of $\Pi(G)$. For each edge e in $\Pi(G)$ we have an edge e'_G in G' connecting the nodes in $\Pi(G)$ corresponding to the two faces that have e on their boundaries.

The dual graphs of a planar graph are known to be equal under 2-isomorphisms [81].

Therefore, there is a one-to-one correspondence between a primal planar graph and its dual graphs because of the inheritance property of 2-isomorphism. For example, connectivity and planarity is preserved. Additionally, cuts in the primal graph correspond to circles in its dual graphs. Expending or deleting, respectively, an edge of the primal graph means deleting or expending, respectively, its corresponding dual edge. For further details the reader is referred to [81].

Planarity of a graph $G = (V, E)$ can be tested in $O(|V|)$ time by, e.g., the algorithm of Hopcroft and Tarjan [49], or an approach of Lempel *et al.* [60] using the special data structure PQ-tree introduced by Booth and Lueker [8]. For a planar graph G, an embedding Π of G can be determined in linear time by, e.g., the algorithms of Chiba *et al.* [14] or Mehlhorn and Mutzel [66].

2.2.2 Cluster Planarity

In Section 2.1.4 we have already discussed clustered graphs and their representation. Here, we adapt the concept of planarity to clustered graphs [30].

A drawing of a clustered graph is *c-planar* if there are no edge crossings or edge-region crossings. If a clustered graph C has a c-planar drawing then we say that it is *c-planar*. Figure 2.6 shows a planar clustered graph that is not c-planar.

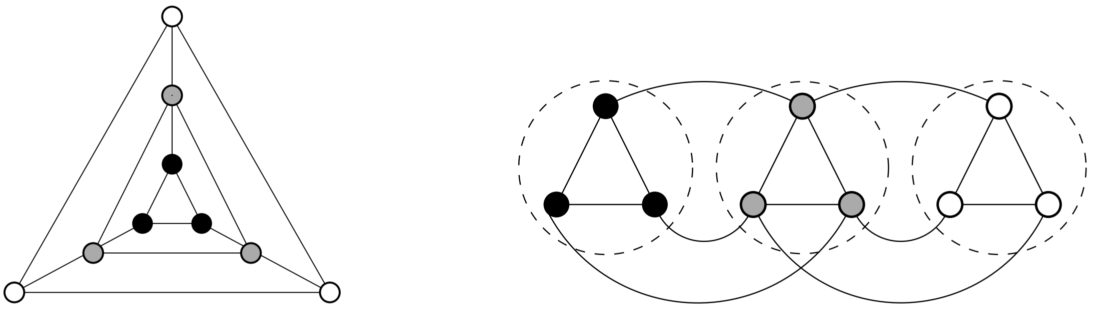


Figure 2.6: A planar clustered graph that is not c-planar [30] (the three disjoint clusters are represented by different types of vertices)

Therefore, a c-planar drawing contains a planar drawing of the underlying graph. An *embedding* of C includes an embedding of G plus the circular ordering of edges crossing the boundary of the region of each non-trivial cluster (a cluster which is not a single vertex).

The following results from [30] characterize c-planarity:

Theorem 2.3. *[30] A c-connected clustered graph $C = (G, T)$ is* c-planar *if and only if graph G is planar and there exists a planar drawing $\mathcal{D}$ of G, such that for each node ν of T, all the vertices and edges of $G - G(\nu)$ are in the outer face of the drawing of $G(\nu)$.*

Theorem 2.4. *[30] A clustered graph $C = (G, T)$ is c-planar if and only if it is a sub-clustered graph of a connected and c-planar clustered graph.*

Theorem 2.5. *[21] A c-connected clustered graph whose underlying graph is a tree is c-planar.*

A further result from [30] is a c-planarity testing algorithm for c-connected clustered graphs based on Theorem 2.3 with running time $O(n^2)$, where n is the number of vertices of the underlying graph and each non-trivial cluster has at least two children. An improvement in time complexity is given by Dahlhaus who constructed a linear time algorithm [18, 20]. Additionally, some special cases are known: If for every cluster ν of a c-connected clustered graph $C_1 = (G, T)$ $G - G(\nu)$ is connected, then C is c-planar if and only if G is planar [15, 54]. The same result holds for bipartite graphs $B = (V_1 \cup V_2, E)$ in which V_1 and V_2 induce each a cluster that are disjoint to each other. Then the corresponding clustered graph $C_2 = (B, T)$ is c-planar if and only if B is planar [30].

2.3 Topological Inference

The following definitions are based on the work of Michelangelo Grigni, Dimitris Papadias and Christos Papadimitriou in [39]:
Of great interest for developing intelligent inference engines for geographic database system are questions like: Given some simply connected regions with relations like A overlaps B and A contains C, what can be said about the relation of B and C?

Egenhofer pointed out in [26] that there exist eight fundamental relations that can hold between two planar regions: `overlap`, `disjoint`, `inside`, `contains`, `meets` (overlaps only at the boundary), `covers` (contains but also shares some boundary), `covered by` (inverse of `contains`) and `equal` (see Figure 2.7). This resolution is called the *high resolution case*. Considering any three planar regions, they cannot be in arbitrary relation. The complete table of such one-step inferences was derived by Egenhofer in [26] and Smith and Park in [77].

In some cases not all relations are needed. E.g., in some cases it makes sense not to distinguish between `covered by` and `inside` resp. between `covers` and `contains` and call both `covered by` and `inside`. Additionally, in some cases it is reasonable not to distinguish between `overlap` and `meets` and to call both `overlap`. This is the *medium resolution case*. In a sub-case of the medium resolution case the relation `overlap` is forbidden. The following table demonstrates the medium resolution one-step inferences without overlap:

	disjoint	**equal**	**inside**	**contains**	**meets**
disjoint	d ∨ e ∨ i ∨ ct ∨ m	d	d ∨ i ∨ m	d	d ∨ i ∨ m
equal	d	e	i	ct	m
inside	d	i	i	d ∨ e ∨ i ∨ ct ∨ m	d ∨ m
contains	d ∨ ct ∨ m	ct	ct ∨ e ∨ i	ct	ct ∨ m
meets	d ∨ ct ∨ m	m	m ∨ i	d ∨ m	d ∨ e ∨ i ∨ ct ∨ m

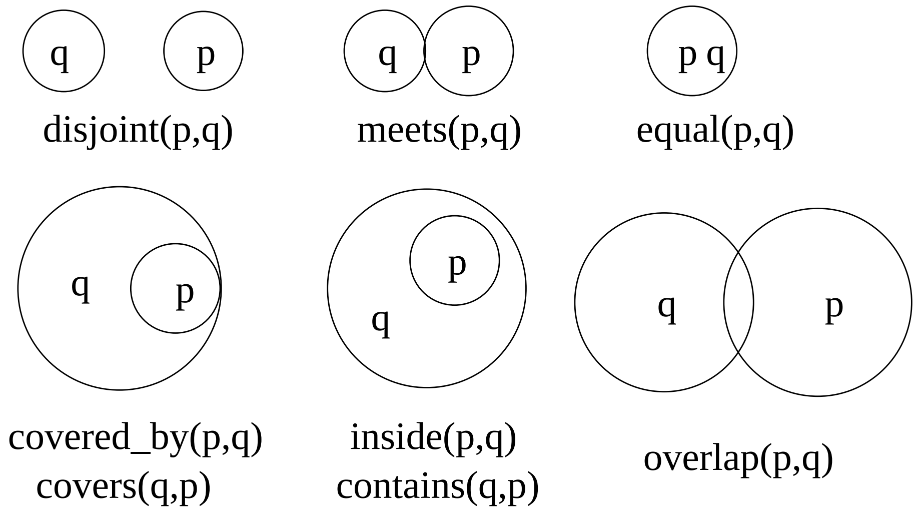


Figure 2.7: The high resolution case of topological relations

The *low resolution case* considers only the relations `overlap` and `disjoint`.

The problem of recognizing whether given pairwise topological relations of a given set of regions are consistent can be seen as a constrained satisfiability problem (for more details see [39]). There is a classification depending on the given topological relations of pairs of regions. We will focus only on the *explicit case*, in which the topological relations of all pairs of regions are known.

Additionally, there is a stronger notion of satisfiability: the *realizability* as a full form of satisfiability. A topological expression on a set of objects is said to be realizable if it has a *planar model*, that is, if there is a set of simply connected planar regions, one for each object, any two of which are related by a topological relation that is a disjunct of the corresponding clause.

In [39] the complexities of all these cases have been studied.

We will consider the explicit case of the medium resolution one-step inferences without overlap realizability problem (EMO) that was first claimed to be in $\mathcal{P}$ in [39] but then revised to be still open.

2.3.1 Planar Topological Inference/ Hierarchical Topological Inference

The following definitions are based on the work of Zhi-Zhong Chen and Xin He in [12, 13]: An instance of the *planar topological inference* (PTI) problem is a triple $(V, \bowtie_d, \bowtie_m)$, where V is a finite set, $\bowtie_d$ (subscript d stands for *disjoint*) and $\bowtie_m$ (subscript stands for *meets*) are two irreflexive symmetric relations on V with $\bowtie_d \cap \bowtie_m = \emptyset$. The problem is to determine whether we can draw the elements v of V in the plane each as a closed disc homeomorph D_v in such a way that

1. D_v and D_w are disjoint for every $(v, w) \in \bowtie_d$, and
2. D_v and D_w have disjoint interiors but share a point of their boundaries for every $(v, w) \in \bowtie_m$.

In the *fully-conjective* case of the PTI problem, it holds that for every pair of distinct $v, w \in V$, either $\{(v, w), (w, v)\} \subseteq \bowtie_d$ or $\{(v, w), (w, v)\} \subseteq \bowtie_m$. This problem is known to be in $\mathcal{P}$. A restriction of the fully-conjective PTI problem, called the fully-conjective k-PTI problem, has been investigated in [12] by Chen, He by requiring that no point of the plane is shared by more than k closed disc homeomorphs D_v with $v \in V$. Notice that the fully-conjective 3-PTI problem is equivalent to deciding whether a given graph is planar.

A generalization is obtained of the PTI problem by adding another irreflexive, antisymmetric and transitive relation $\bowtie_i$ (subscript i stands for *inclusion*) on V with $\bowtie_i \cap (\bowtie_d \cup \bowtie_m) = \emptyset$ and requiring that D_v includes D_w as a sub-region for all $(v, w) \in \bowtie_i$. A natural restriction on $\bowtie_i$ is to require that each D_v with $v \in V$ and $\{(v, w) \in \bowtie_i \mid w \in V\} \neq \emptyset$ is the union of all D_w with $(v, w) \in \bowtie_i$. This generalization is called the *hierarchical topological inference* (HTI) problem. In the *fully-conjective* case of the HTI-problem, for every pair of distinct $v, w \in V$, exactly one of the following holds:

1. $\{(v, w), (w, v)\} \subseteq \bowtie_d$,
2. $\{(v, w), (w, v)\} \subseteq \bowtie_m$,
3. $\{(v, w), (w, v)\} \cap (\bowtie_d \cup \bowtie_m) = \emptyset$ and $|\ \{(v, w), (w, v)\} \cap \bowtie_i\ | = 1$.

Analogously to the PTI case, a restriction called the *fully-conjective k-HTI problem* is derived by adding the following condition: no point of the plane is shared by more than k minimal closed disc homeomorphs D_v, i.e., those D_v with $v \in V$ such that $\{w \in V \mid (v, w) \in \bowtie_i\} = \emptyset$. This problem is known to be in $\mathcal{NP}$ but not known whether it is $\mathcal{NP}$-complete or in $\mathcal{P}$.

2.3.2 Planar Map/ Hierarchical Map

The following definitions are based on the work of Zhi-Zhong Chen and Xin He [12, 13]: A *hierarchical map* $\mathcal{M}$ is a pair $(\mathcal{E}, \mathcal{F})$, where

1. $\mathcal{E}$ is a plane graph whose connected components are biconnected, and
2. $\mathcal{F}$ is a rooted forest whose leaves are distinct faces of $\mathcal{E}$.

Faces of $\mathcal{E}$ that are not leaves of $\mathcal{F}$ are called the *lakes* on $\mathcal{M}$, while all other faces of $\mathcal{E}$ are called the *leaf districts* on $\mathcal{M}$. For each non-leaf vertex α of $\mathcal{F}$, the union of the faces that are leaves in the subtree of $\mathcal{F}$ rooted at α is called a *non-leaf district* on $\mathcal{M}$.
$\mathcal{M}$ is a planar map if each vertex of $\mathcal{E}$ is incident to at most three edges of $\mathcal{E}$. $\mathcal{M}$ is a disc map if

1. the boundary of each leaf district on $\mathcal{M}$ is a cycle, and
2. for every non-leaf district $\mathcal{D}$ on $\mathcal{M}$, there is a cycle C such that $\mathcal{D}$ is the union of the faces in the interior of C.

The *map graph* G of $\mathcal{M}$ is the simple graph whose vertices are the leaf districts on $\mathcal{M}$ and whose edges are those $\{f_1, f_2\}$ such that the boundaries of f_1 and f_2 intersect (this can be more than once). Notice that G is planar when $\mathcal{M}$ is a planar map. $(G, \mathcal{F})$ is the *abstract* of $\mathcal{M}$.
The *graph-forest* pair is a pair of a simple graph G and a rooted forest $\mathcal{F}$ where leaves are exactly the vertices of G. A graph-forest is called planar when G is planar. Notice that a graph-forest is equal to a clustered graph without the root cluster. Chen and He [13] developed an $O(|\mathcal{F}| + |G| \log |G|)$-time algorithm using SPQR-trees (see next section) for deciding whether a graph-forest pair $(G, \mathcal{F})$ is the abstract of a planar disc map $\mathcal{M}$, where $|G|$ denotes the total number of edges and vertices of a simple graph G. The *planar disc embedding problem* is defined as follows: Given a planar graph-forest $(G, \mathcal{F})$, decide whether there is a planar embedding Π of G satisfying the inclusion relation induced by $\mathcal{F}$. This is equivalent to deciding whether a clustered graph is c-planar. Chen and He [13] have given an $O(n \log n^2)$-time algorithm for a special case in which each subgraph induced by a node of $\mathcal{F}$ is connected, called the *connected disc embedding* (CDE) problem.

2.4 SPQR-tree

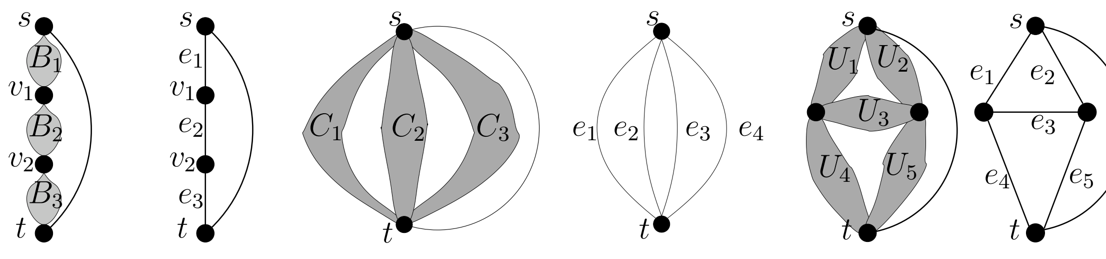
SPQR-trees have been introduced by Di Battista and Tamassia [22]. They represent a decomposition of a planar biconnected graph according to its split pairs (pairs of vertices whose removal splits the graph or vertices connected by an edge). The construction of the SPQR-tree works recursively. At every node v of the tree, we split the graph into the split components of the split pair associated with that node. The first split pair of the decomposition is an edge of the graph and is called the *reference edge* of the SPQR-tree. We add an edge to each of them to make sure that they are biconnected and continue by computing their SPQR-tree and making the resulting trees the subtrees of the node used for the splitting. Every node of the SPQR-tree has two associated graphs:

- The *skeleton* of the node associated with a split pair p is a simplified version of the original graph where some split-components are replaced by single edges.

- The *pertinent graph* of a node v is the subgraph of the original graph that is represented by the subtree rooted at v.

The two vertices of the split pair that are associated with a node v are called the *poles* of v. There are four different node types in an SPQR-tree (S-,P-,Q- and R-nodes) that differ in the number and structure of the split components of the split pair associated with the node. The Q-nodes form the leaves of the tree, and there is one Q-node for each edge in the graph. The skeleton of a Q-node consists of the poles connected by two edges. The skeletons of S-nodes are cycles, while the skeletons of R-nodes are triconnected graphs. P-node skeletons consist of the poles connected by at least three edges. Figure 2.8 shows examples for skeletons of S-, P- and R-nodes.

Skeletons of adjacent nodes in the SPQR-tree share a pair of vertices. In each of the two skeletons, one edge connecting the two vertices is associated with a corresponding edge in the other skeleton. These two edges are called *twin edges*. The edge in a skeleton that has a twin edge in the parent node is called the *virtual edge* of the skeleton.



(a) Case S-node (b) Case P-node (c) Case R-node

Figure 2.8: The structure of biconnected graphs and the skeleton of the root of the corresponding SPQR-tree.

Let $e = (u, v)$ be an edge in a skeleton $\mathcal{S}$ of a node μ of the SPQR-tree $\mathcal{T}$ of G. Since e is an edge in a skeleton, the vertices u and v are a split-pair of G. Then we define the *expansion graph* of e as follows:

1. If μ is the Q-node for edge e' in G, then the expansion graph $G(e)$ of e is defined as follows: if e is the virtual edge of μ then we define $G(e)$ as $G(e) = (V, (E - \{e'\}) \cup \{e\})$. Otherwise, we define $G(e)$ as $G(e) = (\{u, v\}, \{e, e'\})$. Therefore the expansion graph is either isomorphic to G or to $\mathcal{S}$.

2. If μ is an R-node or S-node, then the expansion graph of e is the union of all the split components of the split pair $\{u, v\}$ in G that contain no vertices of $\mathcal{S}$ except u and v together with edge e.

3. If μ is a P-node, then there are at least three split components of the pair $\{u, v\}$ in G. In the construction of $\mathcal{T}$, all edges e_i of $\mathcal{S}$ except the virtual edge are associated with a subgraph G_i of G. Thus we define the expansion graph of each e_i as the graph

G_i together with edge e. The expansion graph of the virtual edge is defined as the split component of $\{u, v\}$ that contains the reference edge of $\mathcal{T}$ (the edge of G that is used to start the decomposition) together with edge e.

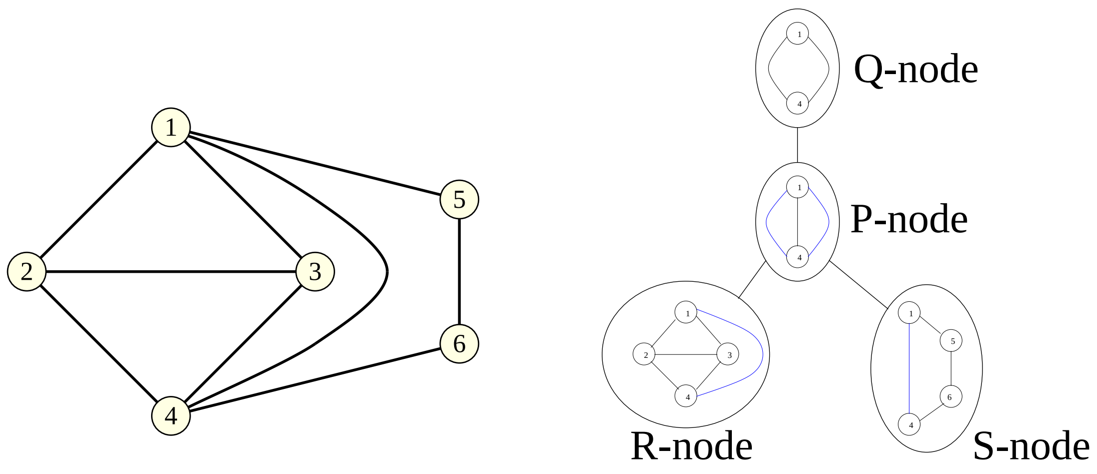


Figure 2.9: A graph G and its SPQR-tree (the Q-nodes of the R- and S-node are omitted).

All leaves of the SPQR-tree are Q-nodes and all inner nodes S-, P- or R-nodes. When we see the SPQR-tree as an unrooted tree, then it is unique for every biconnected planar graph. Another important property of these trees is that their size (including the skeletons) is linear in the size of the original graph and that they can be constructed in linear time [22] and [42]. As described in [22] and [42], SPQR-trees can be used to represent the set of all combinatorial embeddings of a biconnected planar graph. Every combinatorial embedding of the original graph defines a unique combinatorial embedding for each skeleton of a node in the SPQR-tree. Conversely, when we define an embedding for each skeleton of a node in the SPQR-tree, we define a unique embedding for the original graph. The skeleton of S- and Q-nodes are simple cycles, so they have only one embedding. But the skeletons of R-and Q-nodes have at least two different embeddings. Therefore, the embeddings of the R- and P-nodes determine the embedding of the graph and we call these nodes the *decision nodes* of the SPQR-tree.

Part I

Planarity and Planar Augmentation

Chapter 3

State of the Art for Clustered Graphs

In this chapter we summarize the existing methods for drawing clustered and compound graphs. Let $C = (G, T)$ be a clustered graph.

We assume that C is c-connected. Obviously, we may assume that C is planar. In the following we discuss the first c-planarity testing algorithm presented by Feng [30, 29], already mentioned in the Chapter 2 and based on Theorem 2.3.

The idea of the c-planarity testing algorithm is that for each cluster ν it is first checked whether $G - G(\nu)$ can be embedded outside of $G(\nu)$. In the positive case, $G(\nu)$ is replaced by a smaller representative graph that preserves all feasible embeddings of $G(\nu)$. This is done bottom-up in the clustertree T in postorder traversal for each cluster ν in G: the possible embeddings of $G(\nu)$ are found by combining all possible embeddings of the children of ν which are found recursively and are saved for the further testing of the parent of ν. Finally, after reaching the root cluster the resulting graph G is tested for planarity to ensure that such a cluster representation is feasible for C in total.

In particular, bottom-up in postorder traversal in T we check if there is a planar embedding for $G'(\nu)$ that is $G(\nu)$ with the end vertices of its cluster incident edges that belong to $G - G(\nu)$ shrinked in a single dummy vertex.

In the positive case we replace $G(\nu)$ by its representive graph that is a wheel graph. The wheel graph is build of wheels which have a simple cycle called the rim and a center vertex named hub such that each vertex on the rim is connected with the hub by a single edge. In a planar embedding of such a wheel all faces are in a triangular form except the rim face. If the rim face is unbounded in the planar embedding we say that the wheel is given in *canonical form*. Moreover, the wheels share pairwise at most one vertex in the wheel graph. Observe that this vertex corresponds to a cut vertex while the wheel itself represents a block in $G(\nu)$.

For the planarity testing step we use an adaption of the approach for general graphs. To understand the adaption we first investigate briefly in the general approach.

The planarity testing algorithm test planarity for each block of the graph. The vertices of a

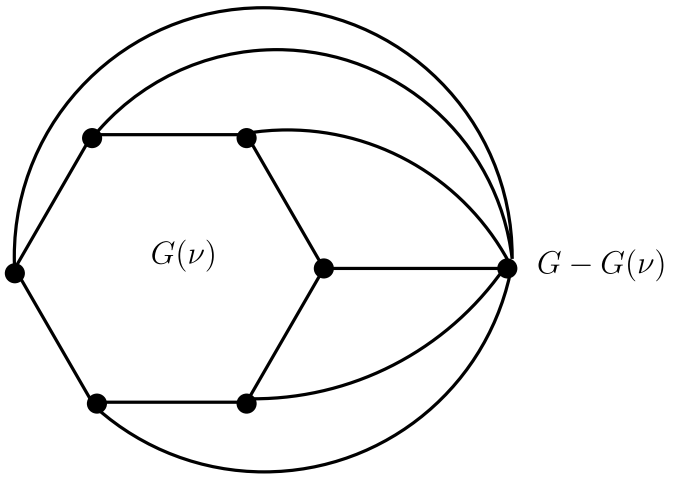


Figure 3.1: c-Planarity Testing

block get a topological numbering, the so-called st-numbering with the following property. Given an edge (s,t) where s is the source with st-number 1 and t is the sink with the number $|V|$ all other vertices are adjacent to both higher and lower st-numbered vertices. The algorithm tests planarity using the vertex addition approach that adds a vertex after each other in st-number ordering starting with the source and ending when all vertices are visited. The correctness of this algorithm follows from the fact that by assigning directions to the edges according to the st-numbering we transform the graph in a dag. Whenever the edge (s,t) of the dag can be embedded in the unbounded face and for every subgraph $G_k = (V_k, E_k)$ of the dag also all edges and vertices of $G - G_k$ can be places in the unbounded face for $k = 1, 2, 3, \ldots, |V|$ then the original graph is planar. To make this process more efficient in its running time, the well-known data structure PQ-tree [8] is used to represent all possible planar embeddings of each G_k from $k = 1$ to $|V|$.

A PQ-tree is based on a set S of elements that form its leaves and has two kinds of internal nodes, the P-node and the Q-node (see Figures 3.2 and 3.3). A P-node allows all possible permutations of its child nodes and a Q-node forbids all permutations except mirroring of the ordering of its children.

We assume that all leaves are horizontally aligned at the bottom, the root at the top, the children of the P- and Q-nodes are ordered and the drawing is planar. Then each such drawing corresponds to exactly one ordering of S, called the *frontier* of T.

Consequently, a PQ-tree gives all possible permutations of its leaves under the given restrictions of its internal nodes. Moreover, for any subset S' of S, we call S' the *restriction set*, we can test whether there exists a frontier with the elements of S' ordered consecutively.

In the positive case, we can even transform the PQ-tree such that the elements of S' are ordered consecutively in *every* frontier of the PQ-tree. In our situation where we apply the vertex addition approach, instead of using G_k, we use the corresponding PQ-tree such

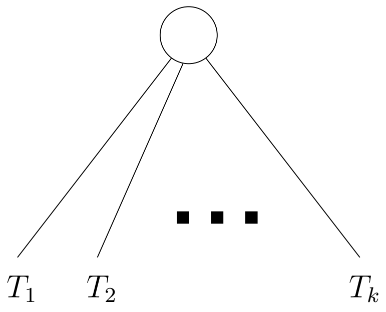


Figure 3.2: A P-node

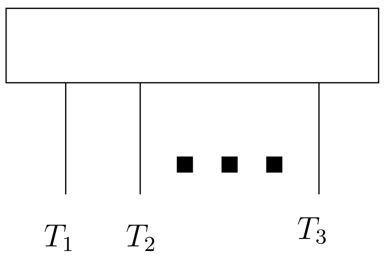


Figure 3.3: A Q-node

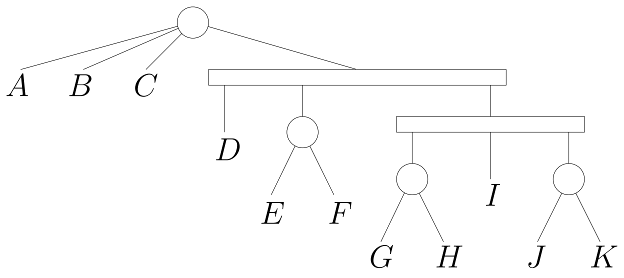


Figure 3.4: A PQ-tree with the set $S = \{A, B, C, D, E, F, G, H, I, J, K\}$

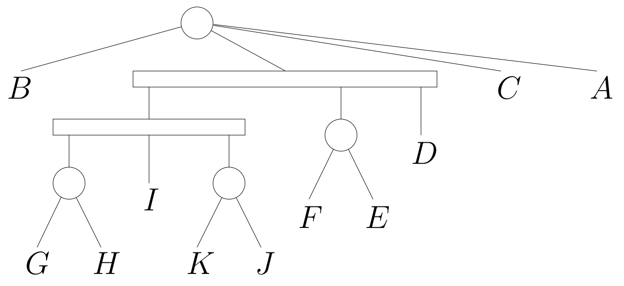


Figure 3.5: A PQ-tree that is isomorph to the PQ-tree of Figure 3.4

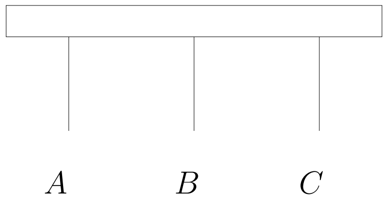


Figure 3.6: A PQ-tree with set $S' = \{A, B, C\}$ and the feasible permutations {ABC,CBA}.

that the P-nodes represent the cut vertices and the Q-nodes the blocks of G_k. The leaves of the PQ-tree correspond to the edges that have an end vertex with st-number higher than k. Next, the PQ-tree is manipulated such that the leaves with label $k+1$ are ordered consecutive in every frontier if such an ordering exists. In the negative case we stop because the graph cannot be planar. If the final vertex with st-number $|V|$ is reached, then G is planar.

Next we turn to the adaption of this planarity testing algorithm to use it for c-planarity testing of c-connected clustered graphs. To test planarity of $G'(\nu)$ for each cluster we apply the PQ-tree on the block that has included the incident edges of ν. To be more precise, the restriction set of the PQ-tree correspond to the incident edges of ν. Whenever this restriction set can be ordered consecutively, we build the wheel graph out of the resulting PQ-tree: for each Q-node we add a wheel and for each P-node we have a vertex on the rim of the wheel graph whose corresponding Q-node is adjacent to the P-node. Observe that this vertex is a cut vertex in the constructed wheel graph that is the representative graph of $G(\nu)$.

Out of this testing algorithm an embedding algorithm is derived by Feng [29, 30]. The main idea is to embed all wheels in its canonical form such that they can be replaced by the original subgraphs and do some post-processing steps to guarantee that $G - G(\nu)$ is embedded outside of $G(\nu)$. The implementation of the algorithm is quite technical and omitted here.

An improvement to linear running time was given by Dahlhaus [18, 20]. Another test was derived by He, Zhi-Zhong [13] that has running time $O(n \log(n^2))$ and was constructed for topological inferences purposes (see Chapter 4). Lengauer [61] present a contribution on hierarchical graphs that also can be adapted for c-connected planar clustered graphs. Additionally, for some other special classes of clustered graphs polynomial time algorithms for testing c-planarity are derived [3, 21, 30, 38, 51].

Cornelson and Wagner [15] state that a c-connected graph for which $G - G(\nu)$ is connected for each cluster ν is c-planar if and only if its underlying graph is planar. This was observed independently by Jünger, Leipert, P. [54] and used in an approach for triangulating clustered graphs that is described in Chapter 8.

Cortese et al. [16] develop an algorithm to test c-planarity of a clustered graph whose underlying graph is a cycle and has a linear running time.

Biedl et al. [6] study planar graphs where each vertex is assigned to one of two disjoint classes. They show a linear time algorithm to test if one of such graphs has a planar drawing such that the vertices of the two classes are separated by an horizontal line, the so-called y-monotone HH-drawing. This can be interpreted as a c-planarity testing of a graph with exactly two sibling clusters at the same level. They also show that a planar bipartite graph has always a y-monotone HH-drawing.

It is a well-known fact that a simple force-directed method can be derived for each cluster ν by simply connecting a dummy vertex to all vertices of ν and assigning hight attracting forces to the added edges.

Eades, Feng and Lin [24] construct an algorithm that produces a straightline convex cluster drawing of a c-connected c-planar embedded clustered graph in $O(n^2)$ running time. Nagamocchi and Kuroya [68] improve the running time when T is a binary tree and each cluster of C is biconnected.

Two orthogonal methods for clustered graphs were derived. The first one is based on the visibility approach and derived by Eades, Feng [25, 29]. The second one is based on the topology-shape approach and is an adaption of Tamassia's algorithm [80].

Multilevel Visualization of clustered graphs is studied in [29].

A planarization approach is given for non c-planar c-connected clustered graphs in [21].

Chapter 4

Complexity of Deciding Compound Planarity

It is easy to see that the general explicit case of the medium resolution one-step inferences without overlap realizability problem can be transformed in polynomial time into an explicit case of the medium resolution one-step inferences without overlap/equal realizability problem (EMOE).

We show a polynomial reduction from the compound planarity problem to the EMOE problem. Let an instance of a compound graph $C = (G, T)$ be given. We transform this instance to an EMOE instance in the following. Then we show that the EMOE instance is realizable if and only if C is compound planar.

We add for each node v of T other than the root a simply connected region R_v, and for each edge e of G a simply connected region R_e.

For each pair of regions R_x, R_y:

1. ***inside*** **and** ***contains*****:**
   Since T is a tree there exists a unique path P_z from each leaf z in T to the root r - see Figure 4.1.

   Since we have a linear number of vertices v in T we have a linear number of leaves and therefore a linear number of such paths P_v for every leaf v of T. Further, each node $v \neq w \neq r$ in such a path P_v for a leaf v has at least one ancestor and at least one descendant in P_v. We visit for every leaf v its unique path P_v: for each node $v \neq w \neq r$ in P_v we add for each descendant w_1

$$R_{w_1} \textit{ inside } R_w$$

$$R_w \textit{ contains } R_{w_1}$$

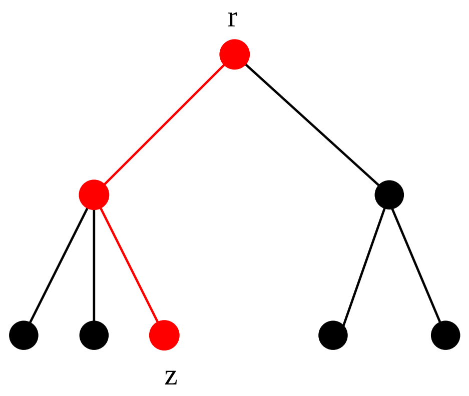


Figure 4.1: In a tree there is a unique path from each leaf to the root, the unique path from leaf z to the root r is visualized red

and for each ascendant w_2 other than the root

$$R_w \textit{ inside } R_{w_2}$$

$$R_{w_2} \textit{ contains } R_w$$

For v itself we add for each ascendant w_2 other than the root in P_v

$$R_v \textit{ inside } R_{w_2}$$

$$R_{w_2} \textit{ contains } R_v$$

In Figure 4.2 the procedure is visualized for an example.

Next, we consider each edge e of graph G. An edge $e = (v, w)$ is related in an inclusion relation to each vertex ν of T that is on the unique path $P(v, w)$ between the lowest common ancestor of v, w and the root of T (see Figure 4.3).

Firstly, we do a pre-processing step to calculate the lowest common ancestors *lca* of each edge $e = (v, w)$ in C. This can be done in linear time in the number of nodes of T [45, 76] and therefore linear in the number of vertices of G. Then, *lca* of an edge e, say $lca(e)$, can be determined in $O(1)$ time [45, 76].

For every edge $e = (v, w)$ we add for each vertex x in $P(v, w)$ other than the root

$$R_e \textit{ inside } R_x$$

$$R_x \textit{ contains } R_e$$

Algorithm 1 gives a pseudo-code description.

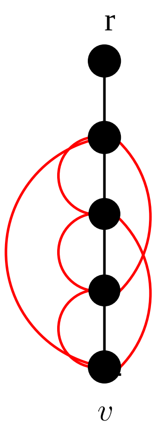


Figure 4.2: A path from a leaf v to the root r of a tree T that is visualized black: for a red edge $e = (v_1, v_2)$ with w.l.o.g v_1 ascendant of v_2 in T we add R_{v_1} *contains* R_{v_2} and R_{v_1} *inside* R_{v_2}

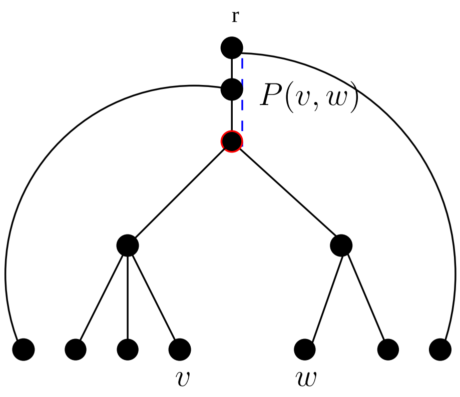


Figure 4.3: A compound tree T: the lowest common ancestors *lca* of leaves v and w is circled red, path from the *lca* to the root r is highlighted with a dashed blue line

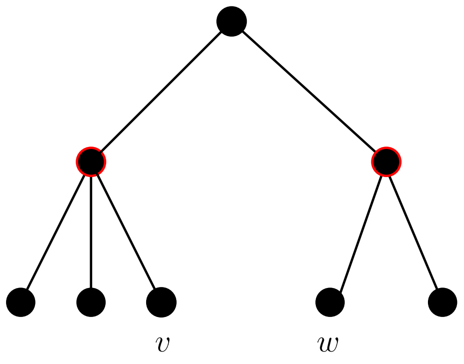


Figure 4.4: A compound tree T: the greatest uncommon ancestors of leaves v and w are circled red

2. ***meet*:**

   Firstly, we do a pre-processing step to calculate the greatest uncommon ancestors *gua* of each edge $e = (v, w)$ in C (see Figure 4.4).

   This can be done in linear time in the number of nodes of T [45, 76] and therefore linear in the number of vertices of G. Then, *gua* of an edge e, say *gua*(e), can be determined in $O(1)$ time [45, 76].

   Given an edge $e = (v, w)$ of G, let *gua*(e)= $\{\nu_1, \nu_2\}$ with $\nu_1 \neq \nu_2$. Further, let ν_1 be an ancestor of v and ν_2 be an ancestor of w in T (see Figure 4.5).

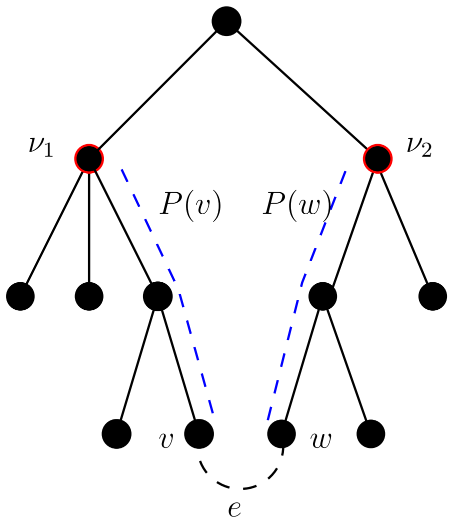


Figure 4.5: A compound tree: for an edge $e = (v, w)$ its greatest uncommon ancestors are ν_1 and ν_2 with paths $P(v)$, $P(w)$.

   Then there is a unique path $P(v)$ from v to ν_1 in T, and a unique path $P(w)$ from w to

ν_2. Observe that e is incident to all nodes on the paths $P(v)$ and $P(w)$. Additionally $P(v) \cap P(w) = \emptyset$. For every edge e, we add

$$R_v \ \textit{meets} \ R_e$$

$$R_e \ \textit{meets} \ R_v$$

$$R_w \ \textit{meets} \ R_e$$

$$R_w \ \textit{meets} \ R_e$$

Additionally, for each inner node x_1 on $P(v)$ and each inner node x_2 on $P(w)$ we add

$$R_{x_1} \ \textit{meets} \ R_e$$

$$R_e \ \textit{meets} \ R_{x_1}$$

$$R_{x_2} \ \textit{meets} \ R_e$$

$$R_e \ \textit{meets} \ R_{x_2}$$

An example is given in Figure 4.6: for each edge $e = (v, w)$ colored red, the previous *meet* relation is assigned to the corresponding regions.

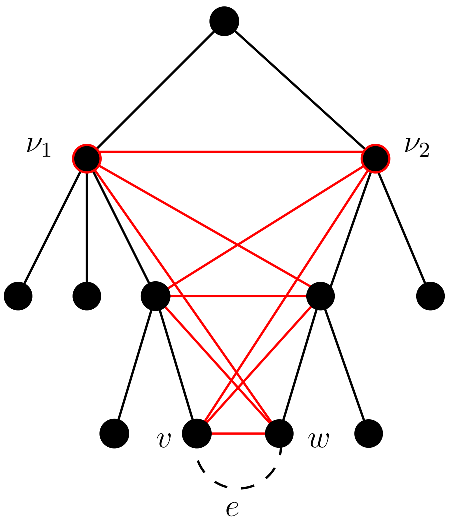


Figure 4.6: For each red edge we add *meet* between the regions that correspond to (v, w) and the regions that correspond to the end vertices of the red edge.

Algorithm 2 gives a pseudo-code description.

3. ***disjoint*:**
   For all other remaining unsigned pairs R_1, R_2 of simply connected regions we add

$$R_1 \ \textit{disjoint} \ R_2$$

$$R_2 \ \textit{disjoint} \ R_1$$

Observe that the constructed EMOE instance fulfills satisfiability in the sense of relational consistency since a compound graph is well-defined. Further, observe that the regions that represent edges of G are pairwise disjoint (this is also true for all regions that correspond to vertices of G that do not stand in any inclusion relation).

We show next that the constructed EMOE instance is realizable if and only if C is compound planar.

1. Given a constructed EMOE instance we show that C is compound planar if the EMOE instance is realizable. If the EMOE instance is realizable there is a realization of the EMOE in the plane such that all simply connected regions are planar and the realization fulfills the satisfiability. We assume that we have such a realization at hand.

   We construct a drawing D of C and show that D is compound planar. Each region that corresponds to an edge $e = (v, w)$ of G stands in a *meet* relation with R_v, R_w and R_{x_1}, R_{x_2} for all vertices $x_1 \in P(v)$, $x_2 \in P(w)$ and is disjoint to any other object. We may assume that R_e has two meeting points with other regions, one with regions R_v, R_{x_1} and the other with R_w, R_{x_2}. Since a simply curve is also simply connected, we replace each region R_e by a simply curve in D that joins the meeting points of R_e with R_v and R_w. Since R_e is planar, the added curve is also planar and therefore does not have any intersection point with another (disjoint) region. For each region that corresponds to a leaf of T we assign the simply connected region as the drawing of v in D. For each inner vertex v in T we assign the border cycle of R_v as the drawing of the boundary of v in D.

   We have in D: the vertices of G are drawn as closed regions so that a vertex u is included in the region representing the vertex parent(u) in T, and the edges in E_G are drawn as curves connecting the regions associated with its end vertices. By definition, we have a compound drawing of C. Additionally, because of the planar model of the realization, we do not have any edge crossings nor any edge-region crossing. Consequently, D is compound planar and therefore C is compound planar.

2. We assume that C is compound planar. Then C has a compound planar drawing D with the properties: the vertices of G are drawn as closed regions so that a vertex u is included in the region representing the vertex parent(u) in T, and the edges in E_G are drawn as curves connecting the regions associated with its end vertices. The only thing that has to be done in D is to shrink the beginning and ending segments in each curve that correspond to $P(v)$ and $P(w)$ according Figure 4.5 (see Figure 4.7).

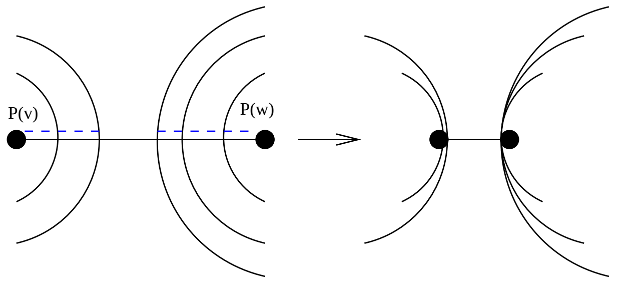


Figure 4.7: Pre-processing step to transform a compound planar drawing into a realization of EMOE: shrink the segments of each curve that correspond to $P(v)$ and $P(w)$ according Figure 4.5

Then, we do not have any edge crossings nor any edge-region crossing. Since the closed regions of the vertices of T and the curves of the edges of G are simply connected, and there is neither an edge crossing nor a edge-region crossing (means all regions are also planar), D correspond to a realization of the corresponding EMOE instance.

Observe that the Algorithms 1 and 2 have in total $O(|V^4|)$ running time, since the greatest uncommon ancestor and the lowest common ancestor, respectively, can be calculated in O(1) running time after a pre-processing step that has $O(|V|)$ running time. Observe that checking if a node in T is an ancestor or descendant, respectively, of a given node of T is included in the algorithm for the greatest uncommon ancestor [45, 76] and has the same running time.

If we assume EMOE to be solvable in polynomial time our polynomial reduction would apply that testing compound planarity, and in particular c-planarity, would be solvable in polynomial time as well. By personal email communications with the authors of [39] the complexity status of EMO and therefore EMOE is still open. Since beside EMO the other realizability problems in [39] are claimed to be $\mathcal{NP}$-hard it seems more likely EMO to be $\mathcal{NP}$-hard as well. However, if testing compound planarity turns out to be $\mathcal{NP}$-hard then EMO is also $\mathcal{NP}$-hard by our polynomial reduction.

Algorithm 1: Algorithm `INSIDE_CONTAINS` assigns *inside* or *contains* to certain pairs of simply connected regions

```
Input: A compound graph $C = (G, T)$, all simply connected regions of EMOE instance
Result: Assignment of *inside* and *contains* to certain pairs of simply connected regions
Pre-processing for $gua$ and $lca$;
Calculate the level of each node in $T$ (root has smallest level);
foreach pair $v$,$w$ of $T$ with $v \neq w$ do
    if pair $v$,$w$ unvisited then
        if $|gua(v,w)| = 1$ then
            if level($v$)>level($w$) then
                Assign $R_v$ *inside* $R_w$;
                Assign $R_w$ *contains* $R_v$;
            else
                Assign $R_w$ *inside* $R_v$;
                Assign $R_v$ *contains* $R_w$;
            Mark pair $v$,$w$ visited;
foreach edge $e = (v, w)$ of $G$ do
    Calculate node $\nu$ of $lca(v,w)$ in $T$;
    Calculate path $P(v, w)$ from $\nu$ to root $r$ in $T$;
    foreach vertex $x$ other than root in $P(v, w)$ do
        $R_x$ *contains* $R_e$;
        $R_e$ *inside* $R_x$;
    Mark pair $v$,$w$ visited;
```

Algorithm 2: Algorithm `MEET` assigns *meet* to certain pair of simply connected regions.

```
Input: A compound graph $C = (G, T)$, all simply connected regions of EMOE
       instance
Result: Assignment of $meet$ to certain pairs of simply connected regions
Pre-processing for $gua$;
foreach edge $e = (v, w)$ of $G$ do
    Calculate nodes $\nu_1$,$\nu_2$ of $gua(v,w)$ in $T$;
    if $\nu_1$ is ancestor of $v$ then
        Calculate path $P(v)$ from $v$ to $\nu_1$ in $T$;
        Calculate path $P(w)$ from $w$ to $\nu_2$ in $T$;
    else
        Calculate path $P(v)$ from $v$ to $\nu_2$ in $T$;
        Calculate path $P(w)$ from $w$ to $\nu_1$ in $T$;
    foreach vertex $v_1$ in $P(v)$ do
        foreach vertex $v_2$ in $P(w)$ do
            $R_{v_1}$ $meets$ $R_e$;
            $R_e$ $meets$ $R_{v_1}$;
            $R_{v_2}$ $meets$ $R_e$;
            $R_e$ $meets$ $R_{v_2}$;
    Mark pair $v$,$w$ visited;
```

Chapter 5

c-Planarity Characterization

In this chapter we give a characterization of c-planar clustered graphs. Since we use duality of planar graphs, the reader is referred to Section 2.2.1 for a short introduction. We adapt the duality for planar clustered graphs in Section 5.1 and show that c-planarity is a facial property in a planar graph in Section 5.2.

5.1 Duality of Planar Clustered Graphs

Given a planar clustered graph $C = (G, T)$ we define its *dual clustering graph* $C^* = (G^*, T^*)$ out of the dual graph G^* and a *clustering relation* T^* with nodes ν^* called *clusterings*:

For each cluster ν in T there is exactly one clustering ν^* in T^* and vice versa.

We assign every edge e^* of G^* to a clustering ν^* of T^* if and only if one of the end vertices of the primal edge e of e^* belongs to ν in C. Additionally, we assign a dual vertex v^* to a clustering ν^* if and only if the corresponding primal face has a vertex of cluster ν on its boundary. The resulting graph is C^*.

We define $I^*(\nu^*) \subset G^*$ to be the dual subgraph of $G(\nu)$ for a cluster ν such that for each vertex v of $G(\nu)$ the dual edges and dual vertices that belong to the boundary of the corresponding dual face of v are contained. An example is given in Figure 5.1. The cluster ν is colored blue while G^* is highlighted with red color. In Figure 5.2(a), all dual faces that correspond to primal vertices of ν are colored blue. $I^*(\nu^*)$ contains all dual edges and dual vertices on the boundary of the blue marked dual faces, as it is shown in Figure 5.2(b) where $I^*(\nu^*)$ in G^* is highlighted with a blue box. Observe that $I^*(\nu^*)$ is the subgraph induced by clustering ν^* in G^*.

We define $H^*(\nu^*) \subset G^*$ to be the dual subgraph of $G - G(\nu)$ for a cluster ν in an analogous way. In Figure 5.3, all dual faces that correspond to primal vertices of $G - G(\nu)$ are colored blue. Hence, all edges on their boundaries belong to $H^*(\nu^*)$. In this example, this is not true for exactly one edge in G^* that is e^*.

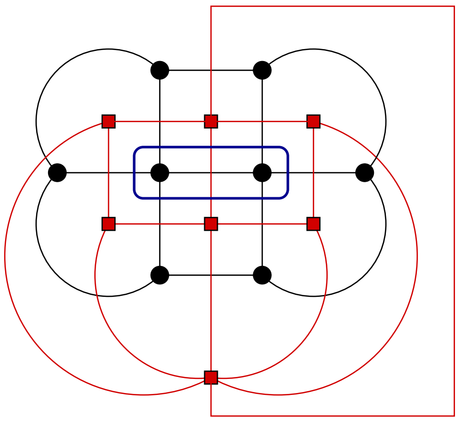
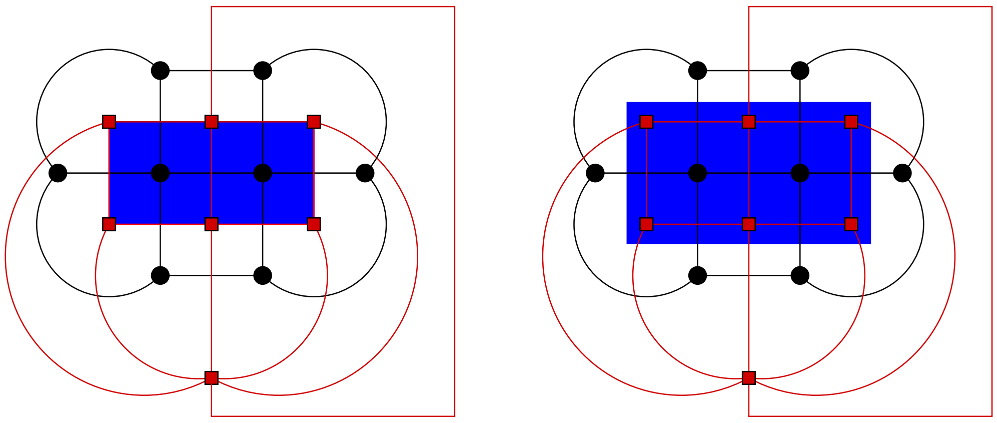


Figure 5.1: Example of a clustered graph, cluster ν is colored blue, the dual of the underlying graph is visualized red



(a) Dual of clustered graph (b) Dual with subgraph $I^*(\nu^*)$

Figure 5.2: Examples of a clustered graph with $I^*(\nu^*)$: on the left-hand side the dual faces that correspond to primal vertices of ν are colored blue while on the right-hand side the dual subgraph $I^*(\nu^*)$ is highlighted with a blue box

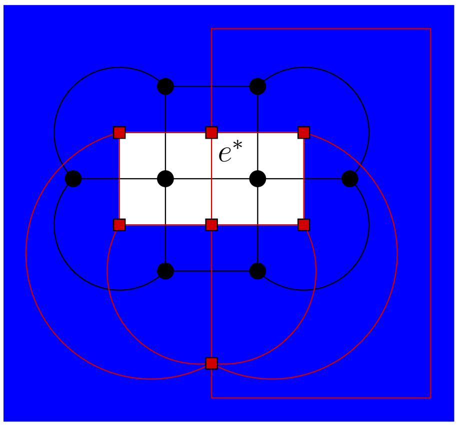


Figure 5.3: Example of a clustered graph with $H^*(\nu^*)$: the dual faces that correspond to primal vertices of $G - G(\nu)$ are colored blue; $H^*(\nu^*) = G^* - e^*$

Observe that C^* is not a clustered graph since there might be edges that do not belong to a cluster ν^* even if its end vertices belong to ν^*. Additionally, even if two primal clusters ν, μ are siblings in T the corresponding dual clusterings ν^*, μ^* in C^* might have common edges and vertices. Furthermore, $I^*(\nu^*) \cap H^*(\nu^*)$ contains all dual edges that correspond to the primal incident edges of cluster ν.

5.2 Facial Characterization of c-Planarity

First, we consider the dual graphs of completely connected (compl-connected for short) or c-connected, respectively, planar clustered graphs (see Figure 5.4).

Observe that primal vertices correspond to faces in the dual graph. In Figure 5.5 a dual face f^* is filled with the same color as the cluster ν that contains the corresponding primal vertex. Notice that the union of the edges and vertices of the boundary of f^* belongs to clustering ν^*. Obviously, a cluster ν is connected if and only if the corresponding dual faces of the vertices of ν are consecutive. In particular, a cluster ν is connected if and only if $I^*(\nu^*)$ is connected. Moreover, the vertices of a cluster ν can be connected in a given planar embedding Γ if and only if $I^*(\nu^*)$ is connected in the dual of Γ.

We say that C^* has a property P under 2-isomorphism if there is a planar embedding of C^* under 2-isomorphism that has property P.

Therefore, we can even straighten our observation: the vertices of a cluster ν can be connected if and only if $I^*(\nu^*)$ is connected under 2-isomorphism.

Let a *cut cluster* be a cluster ν that disconnects G by the removal of $G(\nu)$. We distinguish two types of cut cluster ν: the first type disconnects G^* by removal of $I^*(\nu^*)$ in every dual graph G^* of G. We call the first type the *cycle* cut cluster. The second type does not have

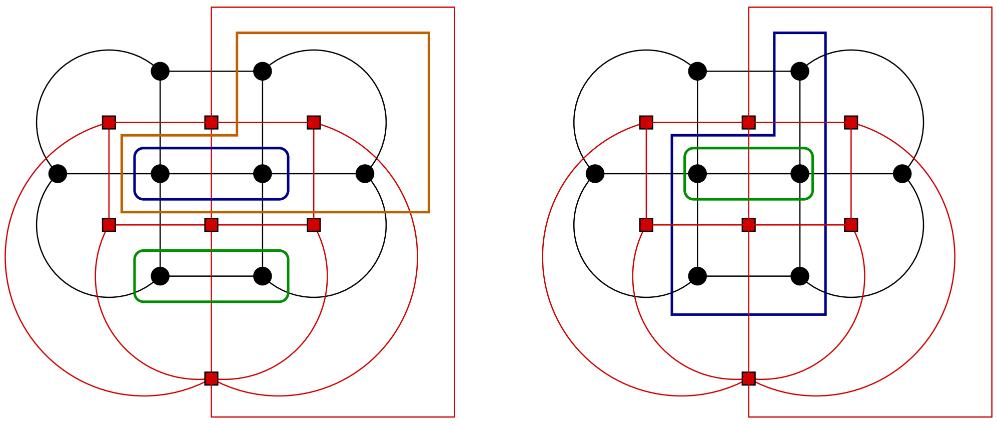

(a) Dual of compl-connected graph (b) Dual of c-connected graph

Figure 5.4: Examples of dual graphs visualized red

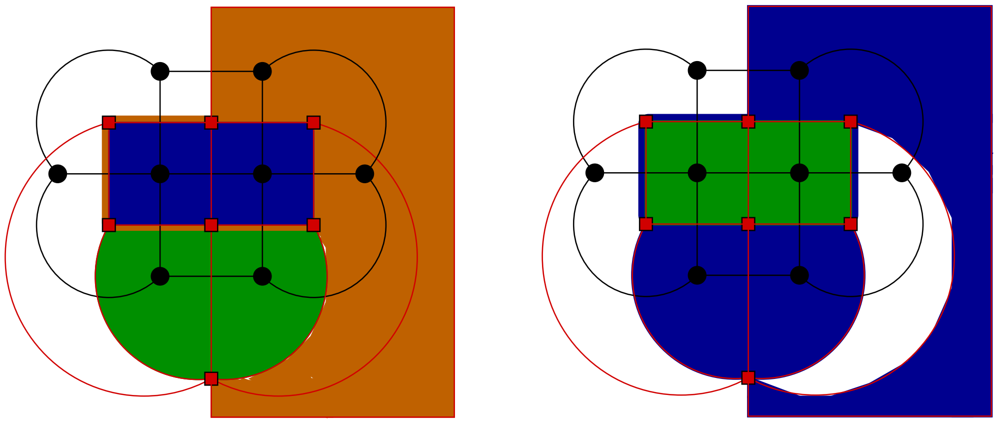

(a) Dual of compl-connected graph (b) Dual of c-connected graph

Figure 5.5: Examples of dual graphs visualized red

this property and hence $H^*(\nu^*)$ is connected under 2-isomorphism (but $I^*(\nu^*)$ might not be connected unter 2-isomorphism).

In Figure 5.5(b) the blue colored cluster form a cut cluster but not a cycle cut cluster.

Recall that in a compl-connected planar clustered graph C for each cluster ν $G(\nu)$ is connected and $G - G(\nu)$ is connected. In other words, for each cluster ν $I^*(\nu^*)$ is connected and $H^*(\nu^*)$ is connected.

5.2.1 c-Connected Clustered Graphs

We can state the following results.

Lemma 5.1. *Let $C = (G, T)$ be a c-connected planar graph and let C^* be its clustering graph. C is c-planar if and only if G has a planar embedding such that for each cluster ν $H^*(\nu^*)$ is connected in its dual G^*.*

Proof. Follows immediately by Theorem 2.3. □

Corollary 5.1. *Let $C = (G, T)$ be a c-connected planar clustered graph. C is c-planar if and only if C does not have a cycle cut cluster.*

Proof. If C does not contain a cycle cut cluster then $H^*(\nu^*)$ is connected under 2-isomorphism for every cluster ν of C. Hence, we can augment C to be co-connected and remain planar. Since C is c-connected C is completely connected and therefore c-planar.

If C is c-planar we can augment C such that it is compl-connected. Then $H^*(\nu^*)$ and $I^*(\nu^*)$ are connected under 2-isomorphism for every cluster ν. Therefore, C does not have a cycle cut cluster. □

Observe that for G being a tree, the previous lemma is trivially satisfied. In Chapter 7 we develop an algorithm that test c-planarity of a c-connected clustered graph based on Corollary 5.1.

5.2.2 co-Connected Clustered Graphs

Recall that we say a clustered graph is co-connected if and only if for every cluster ν $G - G(\nu)$ is connected.

Lemma 5.2. *$C = (G, T)$ is a co-connected planar clustered graph if and only if G is planar and C has no cut cluster.*

Proof. If C is co-connected, then $G - G(\nu)$ is connected for each cluster ν. Hence, C has no cut cluster.
If C has no cut cluster then $G - G(\nu)$ is connected for each non-trivial cluster. Then C is co-connected by definition. □

Since a co-connected clustered graph C has no cut cluster it has no cycle cut cluster. An example is visualized in Figure 5.6.

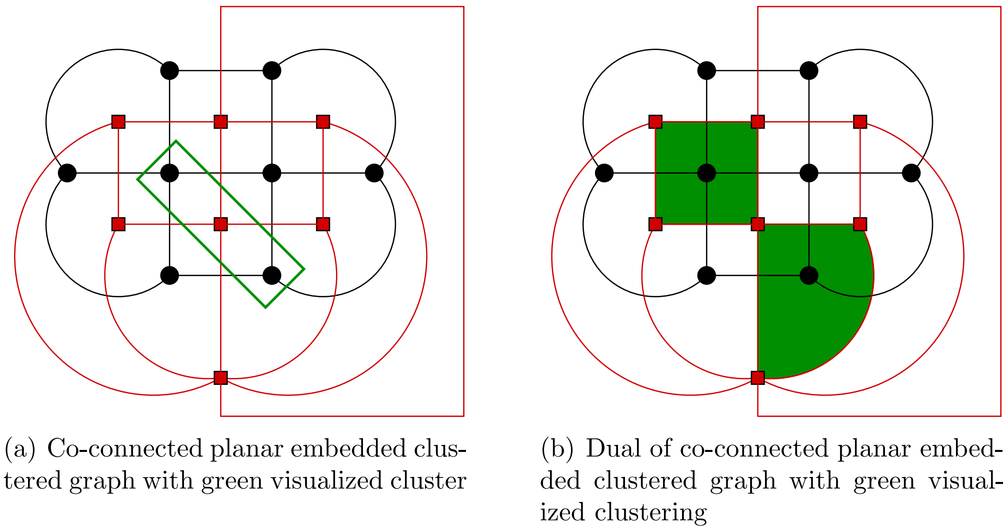

(a) Co-connected planar embedded clustered graph with green visualized cluster
(b) Dual of co-connected planar embedded clustered graph with green visualized clustering

Figure 5.6: Examples of dual graphs visualized red, primal clustered graph has non-connected cluster

Since adding an edge in the primal graph corresponds to expanding an edge in the dual, our example clustered graph of Figure 5.6 turns in the completely connected planar clustered graph visualized in Figure 5.7 by connecting the two vertices of the cluster visualized green.

Observe that $I^*(\nu^*)$ has to be connected under 2-isomorphism to guarantee the successful connectivity augmentation of $G(\nu)$.

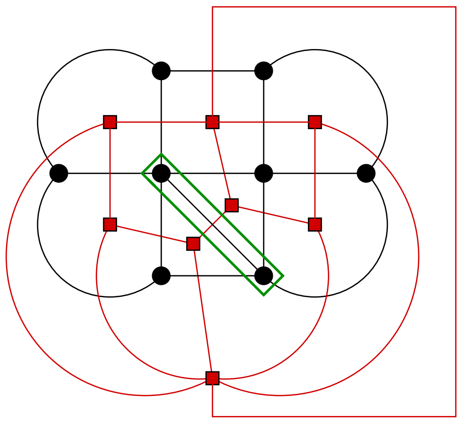
In Figure 5.8 we have a co-connected planar clustered graph C with two disjoint clusters visualized green and blue. We have the situation that we can augment *either* the cluster visualized with blue *or* with green color in C since the clusters are sibling clusters. Let f^* be the dual vertex in C^* placed middle in Figure 5.8. f^* corresponds to the face in the primal planar embedding that has only vertices of both clusters on its boundary. Hence, f^* has to be expanded for both clusters what is impossible. Consequently, C cannot be c-planar.

Since a cluster ν in a co-connected planar clustered graph is not a cut cluster observe that we have to expand every dual vertex that is a cut vertex in $I^*(\nu^*)$ (that itself identify a face in which the primal vertices of $G(\nu)$ are not connected by an edge).

We say that tree X *covers* clustering ν^* if each vertex in ν^* is a vertex in X.

Theorem 5.1. *Let $C = (G, T)$ be a planar co-connected clustered graph. C is c-planar if and only if C^* has the following property P under 2-isomorphism:*

there exists pair-wise non-crossing trees T_{ν^*} such that T_{ν^*} covers ν^* for each clustering ν^* in C^* and for two clusterings ν^*, μ^* with $G(\nu) \subset G(\mu)$ $T_{\nu^*} \subset T_{\mu^*}$.


(a) Co-connected planar embedded clustered graph with green visualized connected cluster

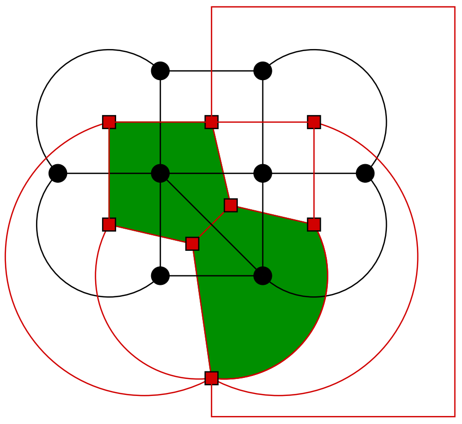
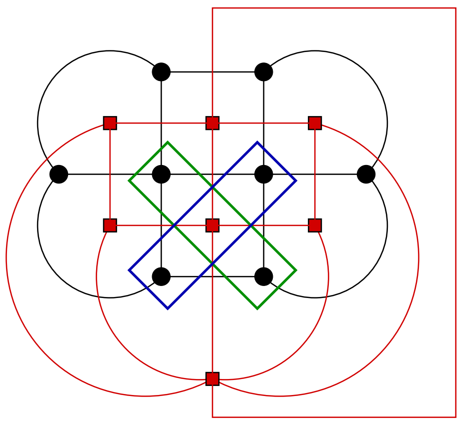

(b) Dual of co-connected planar embedded clustered graph with green visualized connected clustering

Figure 5.7: Examples of dual graphs visualized red, primal clustered graph has connected cluster


(a) Co-connected planar embedded clustered graph with two disjoint clusters visualized green and blue

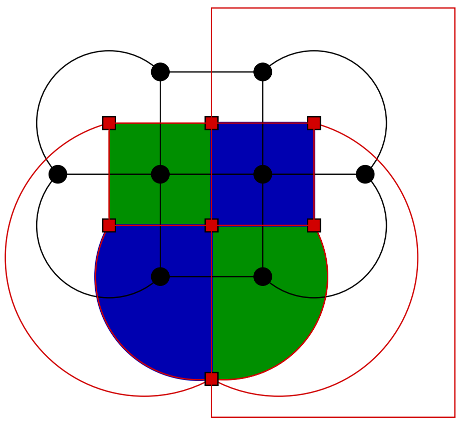

(b) Dual of co-connected planar clustered graph with two clustering visualized green and blue

Figure 5.8: Examples of dual graphs visualized red, primal clustered graph has two disjoint non-connected clusters

Proof. If C is c-planar C can be augmented to a c-connected planar clustered graph. Hence C^* has a planar embedding with the property P under 2-isomorphism.

We assume that C^* has a planar embedding Π^* with property P under 2-isomorphism. Since the trees are non-crossing we can expand each vertex in T_{ν^*} that is a cut vertex in $I^*(\nu^*)$ in the given planar embedding Π^*. Observe that expanding a vertex in the dual graph corresponds to adding an edge in the primal graph. Since T_{ν^*} covers ν^* for each cluster we get a planar embedded c-connected clustered graph. Consequently, since C is also co-connected, C is c-planar. □

5.2.3 General Clustered Graphs

Finally, we consider general planar clustered graphs $C = (G, T)$. We say that tree X *covers* a graph G' if each vertex in G' is a vertex in X.

Since a completely connected planar clustered graph is c-planar [15, 54] we get the following result.

Theorem 5.2. *Let $C = (G, T)$ be a planar clustered graph. For a cluster ν let $\nu_- := G - G(\nu)$ ($\nu_-^* := H^*(\nu^*)$). C is c-planar if and only if C^* has the following property P under 2-isomorphism:*

1. *there exists pairwise non-crossing trees $T_{\nu_-^*}$ such that $T_{\nu_-^*}$ covers ν_-^* for each clustering ν^* in C^* and for two clusterings ν^*, μ^* with $G(\nu) \subset G(\mu)$ $T_{\mu_-^*} \subset T_{\nu_-^*}$ and*

2. *there exists pairwise non-crossing trees T_{ν^*} such that T_{ν^*} covers ν^* for each clustering ν^* in C^* and for two clusterings ν^*, μ^* with $G(\nu) \subset G(\mu)$ $T_{\nu^*} \subset T_{\mu^*}$.*

Proof. Follows immediately by the fact that a c-planar clustered graph has a c-connected and co-connected planar super-clustered graph [15, 54]. □

Observe that each dual vertex corresponds to a primal face. This gives us a primal characterization of c-planarity of a planar graph in terms of facial properties. We use this primal characterization in Chapter 7 where we develop algorithms that test c-planarity in polynomial time for special classes of clustered graphs. The complexity of the general problem remains open.

Chapter 6

Subgraph Induced Planar Connectivity Augmentation

The results of this chapter are joint work with Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel and René Weiskircher and are published in [41].

For an undirected graph $G = (V, E)$, a subset of vertices W of V, and E_W the subset of E that contains only edges with end vertices in W let $G_W = (W, E_W)$ be the subgraph of G induced by W. If G is planar, a *subgraph induced planar connectivity augmentation* for W is a set F of additional edges with end vertices in W such that the graph $G' = (V, E \cup F)$ is planar and the graph G'_W is connected.

We present a linear time algorithm based on the SPQR data structure that tests if a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set. The difficulty of the subgraph induced planar connectivity augmentation problem arises from the fact that the computation of an appropriate planar embedding is part of the problem. Once the embedding is fixed, the decision becomes trivial.

6.1 An Easy Case: Fixed Embedding

We consider the case that the planar graph G is given together with a fixed embedding. In the following, we call the vertices belonging to W *blue vertices*. Our task is to insert edges so that the induced subgraph G_W is connected and G is still planar after edge insertion.

Our algorithm looks at each face f in G that has at least two non-adjacent blue vertices on its boundary (see Algorithms 3-5 for pseudo-codes). We start at an arbitrary blue vertex v on the boundary of f and introduce a new edge through f that connects it to the next blue vertex on the boundary. Thus we step through the blue vertices on the boundary of f, connecting each to its successor on the boundary until we come back to v. We call the resulting graph G'. Note that we only introduced a linear number of edges in this step and that G' is planar. Another important property of G' is that G'_W is connected if and only if there is a planar augmentation for W in G.

Then we compute the graph G'' by deleting all vertices from G' that are not blue. We assign value 1 to all edges introduced in the first step and 0 to all other edges. We can find a minimum spanning tree in G'' in linear time because there are only two different weights on the edges. One way to do this is to use Prim's Algorithm where we use two lists instead of the priority queue. The algorithm may now determine that G'' is not connected. Then we know that there is no planar augmentation for G. Otherwise, the edges of weight 1 in the minimum spanning tree are our solution. Thus, we can solve the problem in linear time.

Algorithm 3: Algorithm `ConnectSubgraphFix` checks if a planar connectivity augmentation in a fixed embedded graph G exists.

```
    Input: A graph $G$, a subset $W$ of vertices in $G$ and an embedding $\Pi$ of $G$
    Result: A planar connectivity augmentation of $G$ if it can be built as an extension
            of $\Pi$ and "false" otherwise
    BuildComponentsGraph;
    return FeasibilityCheck;
```

Algorithm 4: Algorithm `BuildComponentsGraph` that builds graph copies G' of G where the vertex set W' is the set of connected components of the subgraph G_W of G induced by W.

```
    Input: A graph $G$, a subset $W$ of vertices in $G$ and an embedding $\Pi$ of $G$
    Result: Graph copies $G'$ of $G$ with the information about the connected components
            of the subgraph $G_W$
    Copy graph $G$ in graph $G^*$;
    Delete all edges $e = (v, w)$ in $G^*$ with $\neg(v, w \in W)$;
    Detect all connected components $C_i$ $(i \in \mathbb{N})$ of $G^*$;
    Copy graph $G$ in graph $G'$;
    forall the $C_i$ of $G'$ do
        Create list $W_C$ containing all vertices $v \in W \cap C_i$ that correspond to the con-
        nected component $C_i$ of $G^*$;
        Mark all vertices of $W \cap C_i$ with $w_C$ and shrink them to one vertex $w_C$ concerning
        the embedding $\Pi$;
```

The augmented graph G will be returned if there exists one feasible augmentation. Otherwise, the algorithm returns "false".

Algorithm 5: Algorithm `FeasibilityCheck` returns the augmented graph G if the subgraph G_W was successfully connected, false otherwise

```
Input: The graph $G'$, the subsets $W_C$ of vertices in $G'$ and the embedding $\Pi'$ of $G'$
Result: true, if the subgraph $G_W$ is successful connected, false otherwise
forall the faces $f'$ of $G'$ do
    Count the number of vertices of $W_C$ in the boundary of $f'$;
forall the faces $f'$ of $G'$ do
    if the number of vertices of $W_C$ is at least 2 then
        Save $f'$ in a list listcount;
forall the faces $f'$ of listcount do
    Detect the corresponding face $f$ in $G$;
    forall the vertices $v$ in the boundary of $f$ do
        Mark first vertex of the boundary as first;
        Mark the last connected vertex as second;
        Mark the last visited vertex as last;
        if last is not blue marked and $v$ is blue marked and second and $v$ are from
        different connected components then
            Connect second and $v$;
            MergeW;
forall the vertices $v \in W$ do
    if $v$ is not in the last merged component then
        return "false";
```

6.2 The Algorithm for the Biconnected Case

In this section we present an algorithm for the case that the given graph G is biconnected. First, we compute the SPQR-tree $\mathcal{T}$ of G. In Section 6.2.1, we present a recursive algorithm for coloring all edges in all skeletons of the SPQR-tree. This coloring stores information about the position of the blue vertices in each skeleton of the SPQR-tree by assigning three different colors to the edges. The coloring enables us to test if an augmentation is possible by examining the colors in each skeleton. If an augmentation is possible, we compute an embedding of the graph that allows an augmentation (see Section 6.2.2). Then we can apply the algorithm of the previous section to this fixed embedding in order to compute the list of edges needed to solve the augmentation problem. Algorithm 11 gives an overview of the algorithm for biconnected graphs.

6.2.1 The Coloring Algorithm

Again we call a vertex blue, if it is contained in W and black otherwise (see Figure 6.1 and Figure 6.2). We assign one of two colors to each edge in each skeleton: blue or black. We call an edge in a skeleton blue, if its expansion graph contains blue vertices and black otherwise.

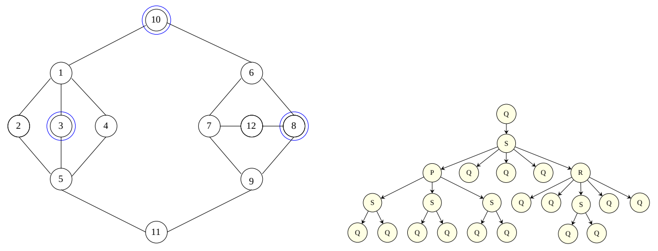


Figure 6.1: Example: A graph G and its SPQR-tree.

Additionally, we assign the attribute *permeable* to some blue edges. Intuitively, an edge is permeable if we can construct a path connecting only blue vertices through its expansion graph. Let $G(e)$ be the expansion graph of edge e in skeleton $\mathcal{S}$. In any planar embedding $G(e)$, there are exactly two faces that have e on their boundary. This follows from the fact that in a planar biconnected graph, every edge is on the boundary of exactly two faces in every embedding. We call the edge e in $\mathcal{S}$ permeable with respect to W, if there is an embedding Π of $G(e)$ and a list of at least two faces $L = (f_1, \ldots, f_k)$ in Π that satisfies the following properties:

1. The two faces f_1 and f_k are the two faces with e on their boundary.
2. For any two faces f_i, f_{i+1} with $1 \leq i < k$, there is a blue vertex on the boundary between f_i and f_{i+1}.

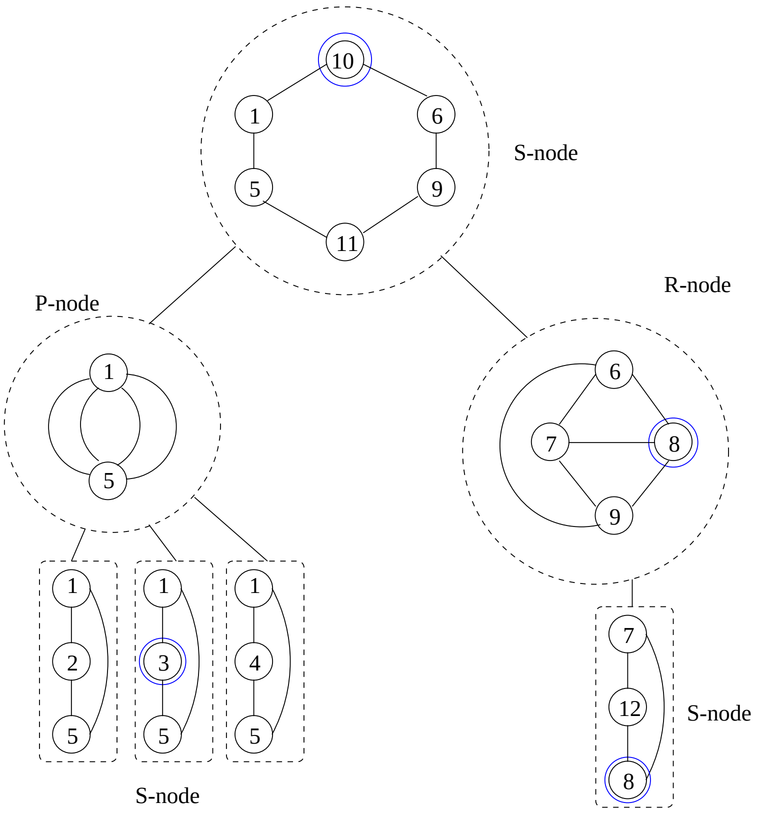


Figure 6.2: Continuation of Figure 6.1: The SPQR-tree where the Q-nodes are omitted; assigning blue to the vertices of the subset W (visualized with a circle around the vertices).

We call a skeleton $\mathcal{S}$ of a node v of $\mathcal{T}$ permeable if the pertinent graph of v together with the virtual edge of $\mathcal{S}$ have the two properties stated above. So $\mathcal{S}$ is permeable if the twin edge of its virtual edge is permeable. We call an expansion graph $G(e)$ permeable if it has an embedding Π and a list of at least two faces $L = (f_1, \ldots, f_k)$ that satisfies the two properties stated above.

We develop an algorithm that marks each edge in every skeleton of the SPQR-tree $\mathcal{T}$ of G with the colors black or blue and that assigns the attribute permeable depending on the expansion graph of the edge. The algorithms works recursively. We assume that $\mathcal{T}$ is rooted at node r and r is not a Q-node.

First we mark all the edges of the skeletons of the children of r recursively black or blue and assign the permeable attribute by treating them as the roots of subtrees. Each edge in the skeleton $\mathcal{S}$ of r except the reference edge corresponds to a child of r. Let e be such an edge in $\mathcal{S}$, v the corresponding child of r and $\mathcal{S}'$ the skeleton of v. If $\mathcal{S}'$ contains a blue edge or vertex, we mark e blue and otherwise black. The permeability of e depends on the type of v:

Q-node: We mark e permeable if the skeleton $\mathcal{S}'$ contains a blue vertex.

S-node: If the skeleton $\mathcal{S}'$ contains a blue vertex or a permeable edge, we mark e permeable.

P-node: If the skeleton $\mathcal{S}'$ contains only permeable edges or a blue vertex, we mark e permeable.

R-node: We consider a graph H where the vertices are the faces of $\mathcal{S}'$ and there is an edge between two vertices if there is a permeable edge or a blue vertex on the boundary that separates the two faces. Let s and t be the two faces left and right of the virtual edge of $\mathcal{S}'$. If there is a path in H connecting s and t, we mark e permeable (see Figure 6.3 and Algorithm 20).

After executing this algorithm, which we call `MarkEdgesPhase1` (see Algorithm 7 for a pseudo-code), all edges of the skeleton of the root node r are marked, because we have visited all the blue vertices of the graph. All other skeletons except the skeleton of r contain one edge that is not yet marked: the virtual edge of the skeleton.

The algorithm `MarkEdgesPhase2` (see Algorithm 22 for a pseudo-code) works top down by traversing $\mathcal{T}$ from the root to the leaves. The edges of the skeleton of the root r of $\mathcal{T}$ are already marked in the first step, therefore we can proceed to the children and mark the virtual edges of its children. Let v be a node in $\mathcal{T}$ where the skeleton $\mathcal{S}'$ of the parent node is already completely marked. We mark the virtual edge e in the skeleton $\mathcal{S}$ of v blue if there is a blue edge or vertex in the skeleton of the parent. The permeability of e again depends on the type of the skeleton in exactly the same way as in the algorithm `MarkEdgesPhase1` (see Figure 6.4). Note that the case Q-node is irrelevant here because the Q-nodes form the leaves of the tree.

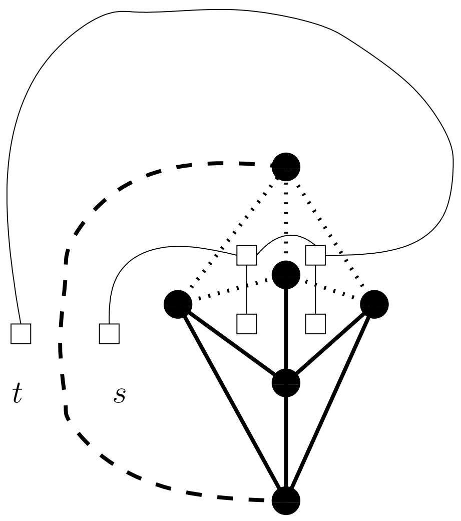


Figure 6.3: A permeable R-node with the graph H: permeable edges are represented by dotted lines, the virtual edge of the skeleton by a dashed line

Algorithm 6: Algorithm `findPath` checks if the expansion graph of the twin edge of an edge in an R-node skeleton is permeable.

```
Input: An edge $e$ and an $R$-node $v$ in $\mathcal{T}$ with the property that $e$ is contained in
       the skeleton $\mathcal{S}$ of $v$ and all edges in $\mathcal{S}$ are marked
Result: "true" if the expansion graph of the twin edge of $e$ is permeable
Compute an arbitrary embedding $\Pi$ of $\mathcal{S}$;
Compute the dual graph $D$ of $\mathcal{S}$ with respect to $\Pi$;
foreach edge $e'$ of $D$ do
    if the primal edge of $e'$ is permeable then
        Set the cost of $e'$ to zero;
    else
        Set the cost of $e'$ to one;
Set the cost of the dual edge of $e$ to one;
Let $v_1$ and $v_2$ be the two vertices in $D$ that correspond to the two faces in $\Pi$ with
$e$ on their boundary;
Compute the shortest path $p$ from $v_1$ to $v_2$ in $D$;
if the cost of $p$ is zero then
    return "true";
else
    return "false";
```

Algorithm 7: Algorithm `MarkEdgesPhase1` that marks all edges in the skeletons except the virtual edges.

```
Input: An node $v$ in $\mathcal{T}$ and the set $W$ of vertices
Result: All edges in $S(v)$ except the virtual are marked.
Let $L$ be the set of children of $v$;
foreach $v' \in L$ do
    MarkEdgesPhase1($v', W$);
    Let $e$ be the edge that $S(v')$ shares with $S(v)$;
    if $S(v')$ contains a blue or permeable edge or a vertex of $W$ then
        Mark $e$ blue
    else
        Mark $e$ black;
    switch type of node $v'$ do
        case $P$-node
            if All edges in $S(v')$ except $e$ are permeable or one of the vertices in $S(v')$
            belongs to $W$ then
                Mark $e$ permeable
        case $S$-node
            if $S(v')$ contains a vertex of $W$ or an edge marked permeable then
                Mark $e$ permeable
        case $R$-node
            Let $e$ be the edge shared by $S(v')$ and $S(v)$;
            if findPath($e, v'$) returns true then
                Mark $e$ permeable
if $v$ is a $Q$-node then
    Let $e$ be the non-virtual edge of $S(v)$;
    if $S(v)$ contains a vertex of $W$ then
        Mark $e$ permeable;
    else
        Mark $e$ black;
```

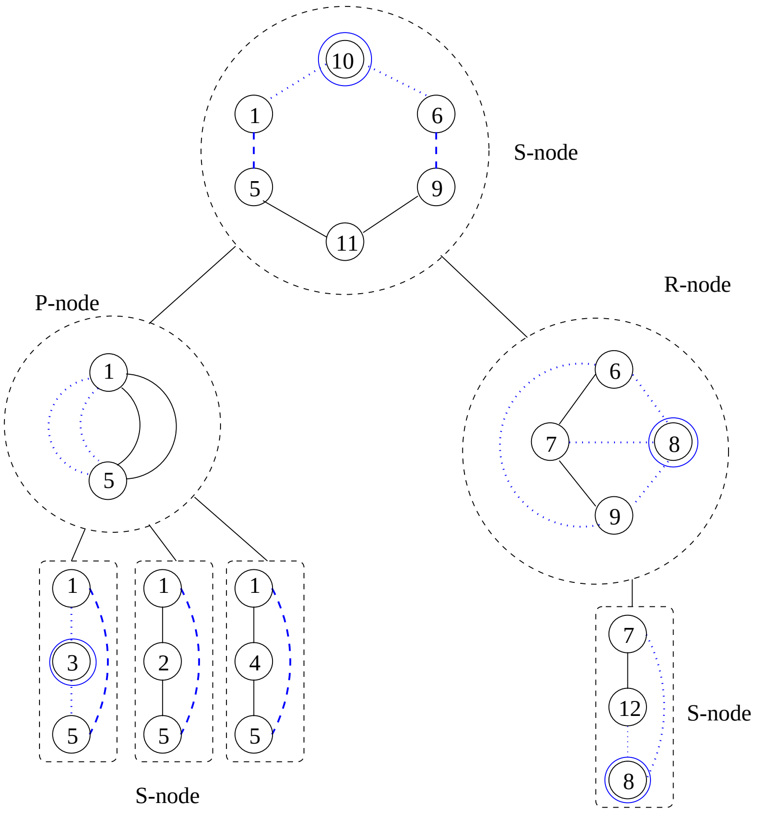


Figure 6.4: Continuation of Figure 6.2: The marking of the edges of the skeletons of the nodes of the SPQR-tree after calling the algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2`; permeable edges are represented by dotted lines, blue edges by dashed lines and blue vertices by a circle around them.

Algorithm 8: Algorithm `MarkEdgesPhase2` marks all the virtual edges of the skeletons in the subtree rooted at v if all the edges in the skeleton of v are marked.

```
Input: A node $v$ in the SPQR-tree $\mathcal{T}$ where all edges are marked
Result: All edges in the subtree rooted at $v$ are marked
Let $\mathcal{S}$ be the skeleton of $v$;
if $v$ is a $Q$-node then
    Let $v_1$ and $v_2$ be the two vertices in $\mathcal{S}$;
    Let $e$ be the virtual edge in the child of $v$;
    if $\{v_1, v_2\} \cap W \neq \emptyset$ then
        Mark $e$ permeable;
    else
        Mark $e$ black;
else
    Let $L$ be the list of edges in $\mathcal{S}$ whose twin edge is contained in a skeleton of a
    child of $v$;
    if $\mathcal{S}$ does not contain a blue or permeable edge or a vertex of $W$ then
        Mark all the twin edges of the edges in $L$ black;
    else
        Mark all the twin edges of the edges in $L$ blue;
        switch type of $v$ do
            case $S$-node
                if $\mathcal{S}$ contains a vertex of $W$ or an edge marked permeable then
                    Mark every twin edge of the edges in $L$ permeable;
            case $P$-node
                if All edges in $\mathcal{S}$ are marked permeable then
                    Mark all twin edges of the edges in $L$ permeable
            case $R$-node
                foreach edge $e$ in $L$ do
                    Let $v'$ be the child of $v$ that contains $e$;
                    if findPath($e, v'$) returns true then
                        Mark $e$ in $S(v')$ permeable;
forall the children $w$ of $v$ do
    MarkEdgesPhase2($w, W$);
```

The two algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2` can both be implemented in linear time because the size of the SPQR-tree of a planar biconnected graph including all skeletons is linear in the size of the graph [22].

Lemma 6.1. *Let e be an edge in a skeleton of an inner node and $G(e)$ its expansion graph. Then the coloring algorithm marks e blue if and only if $G(e)$ contains a vertex of W. Furthermore, e is marked permeable if and only if there is an embedding Π of $G(e)$ together with a sequence of faces $f_1, \dots, f_k$ with the following property:*

(*) The two faces f_1 and f_k are the two faces with e on their boundary and for any two faces f_i, f_{i+1} with $1 \leq i < k$, there is a blue vertex on the boundary between f_i and f_{i+1}.

Proof. The statement is obvious for blue and black edges because if a vertex of W exists in the expansion graph of an edge e, e is marked blue and otherwise black. Hence we have to consider the permeable edges.

We use induction over the number k of inner nodes of the SPQR-tree of the graph $G(e)$. Note that if a graph G is permeable, then $(*)$ holds for every embedding of G. Therefore if we fix the embedding and the property $(*)$ holds then it holds for every embedding and if the property $(*)$ does not hold for the fixed embedding then it will hold for none.

$\mathbf{k = 0}$**:** $G(e)$ consists of two edges and two vertices and the SPQR-tree of $G(e)$ consists of two Q-nodes and an edge. If $G(e)$ contains at least one blue vertex, then it has to be a pole vertex of the skeleton of each of the Q-nodes. Therefore the two faces that have e on its boundary have a blue vertex on its boundary. Therefore e has to be marked permeable. It is obvious that if an edge e represents a Q-node and it is marked permeable then $G(e)$ has to be permeable.

$\mathbf{k > 1}$**:** Let v' be the node attached to v that contains the twin edge of e and S' the skeleton of v'. By induction, the edges in S' are marked permeable if and only if $(*)$ holds. We distinguish three cases defined by the type of v'.

S-node: If S' contains a vertex of W, our algorithms mark e permeable and it is obvious that in this case $(*)$ holds (the blue vertex is on the boundary of the same faces as the virtual edge in S' and therefore on the boundary between the same faces as e in S). If S' contains a permeable edge e'', our algorithms mark e permeable. By induction, we know that there is an embedding Π of the expansion graph of e'' and a sequence of faces such that property $(*)$ holds. It follows that there is an embedding and a sequence of faces in S' such that property $(*)$ holds (e'' is on the boundary of the same two faces as the virtual edge in S' and therefore as e in S) and that $G(e)$ is indeed permeable.

P-node: If S' contains a vertex of W, our algorithms mark e permeable and it is obvious that in this case $(*)$ holds because S' contains only pole vertices. If S' contains only permeable edges, our algorithms mark e permeable. Let $e''_1, \ldots, e''_j$ be a sequence of all permeable edges in S'. By induction, we know that there is an embedding Π for the expansion graph of each permeable edge and a sequence of faces such that property $(*)$ holds. As the permeable edges are incident to its end vertices, we know that every face in S' has two permeable edges on its boundary. Hence we can concatenate the sequences of faces according to the embeddings Π of the expansion graphs in S'. Therefore it follows that there is an embedding and a sequence of faces in S' such that property $(*)$ holds. Hence $G(e)$ is permeable.

R-node: If S' contains a pole vertex that is contained in W, our algorithms mark e permeable and it is obvious that in this case $(*)$ holds because the pole vertex is an end vertex of e. If S' contains a path in the graph H (where the vertices are faces of the skeleton and there is an edge between two vertices if there is a permeable edge or a blue vertex on the boundary that separates the two faces and the vertices that represents the two faces with the virtual edge on their boundary are named s and t) connecting s and t, our algorithms mark e permeable. Let P be such a path in H from s to t. By induction, we know that there is an embedding Π for the expansion graph of each permeable edge and a sequence of faces such that property $(*)$ holds. As such a path exists, then every blue vertex is on a boundary of a face where at least a permeable edge or another blue vertex exists. Hence we can concatenate the sequences of faces according to path P in S'. Therefore it follows that there is an embedding and a sequence of faces in S' such that property $(*)$ holds. Hence $G(e)$ is indeed permeable.

$\square$

6.2.2 The Embedding Algorithm

Let $\mathcal{S}$ be a skeleton of a P-node. We call the embedding of $\mathcal{S}$ *admissible* if all blue edges are consecutive and the blue edges that are not permeable (if they exist) are at the beginning and at the end of the sequence (see Figure 6.5 for three examples of admissible orderings).

Our algorithm for finding an augmentation or proving that no augmentation exists works in two phases:

1. Using the colors and attributes of the edges in each skeleton, we fix an embedding for every P- and R-node skeleton and thus determine an embedding for G.

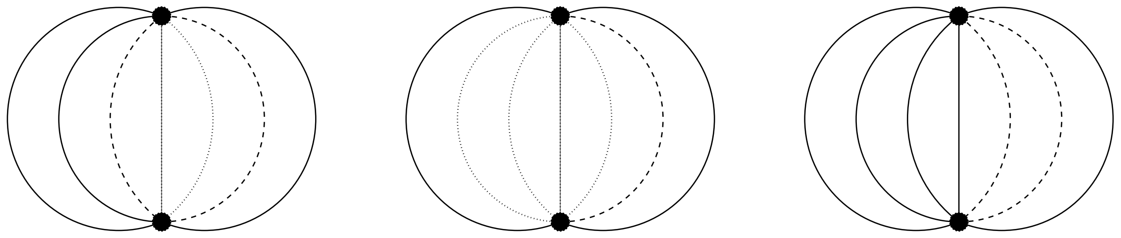


Figure 6.5: Admissible embeddings of a P-node skeleton (permeable edges are dotted, blue edges that are not permeable are dashed)

2. We use the algorithm of Section 6.1 for fixed embeddings to determine whether an augmentation is possible.

The embedding computed in the first step has the property that it allows an augmentation if and only if there is an embedding of G that allows an augmentation.

We set the embedding for the skeletons of the R- and P-nodes recursively using the structure of the SPQR-tree. We assume that the vertices in G are numbered and that all edges are directed from the vertex with lower number to the vertex with higher number.

For simplicity, we consider whether a special case is presented where a planar connectivity cannot exist.

Theorem 6.1. *Let G be a biconnected series-parallel planar graph and W a subset of its vertices. There exists a planar connectivity augmentation for W in G if and only if all P-nodes of the SPQR-tree of G contain at the most two edges that are blue but not permeable.*

Proof. The SPQR-tree of series-parallel graphs contain no R-node. P-nodes with at least three blue and no permeable edges cannot be augmented (Figure 6.6 shows a graph whose SPQR-tree contains a P-node with three blue but not permeable edges) and therefore no planar connectivity augmentation exists.

If every P-node skeleton contains at most two edges that are blue but not permeable, we can always find an embedding of the graph where we can connect the blue vertices. The embedding of G is determined by the embedding of each P-node skeleton. □

The conclusion of the theorem is that whether there exists a P-node that contains a skeleton with more than two blue edges in a biconnected graph G, there cannot exist a planar connectivity augmentation.

First, we test whether the biconnected graph contains only P-nodes with skeletons that have at the most two edges. Then we can sort the two blue edges as mentioned in Figure 6.5 to obtain an admissible embedding.

It is obvious that this can also be done in linear time (see Algorithm 9 for a pseudo-code).

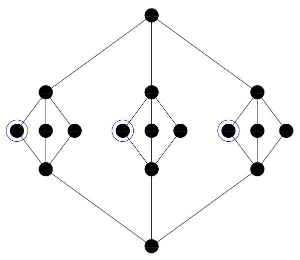


Figure 6.6: Three blue marked nodes in a graph which corresponding P-node has three blue edges; in this situation the subgraph induced by W cannot be connected so that the graph remains planar.

Algorithm 9: Algorithm `BiconnectivityFeasibilityCheck` checks if the skeletons of P-nodes can be augmented.

```
Input: The SPQR-tree $\mathcal{T}$ of $G$ where all edges are marked and the subset $W$ of
       vertices of $G$
Result: “true” if a planar connectivity augmentation for $W$ in $G$ is possible and
        “false” otherwise
result = “true”;
foreach $P$-node $v$ of $\mathcal{T}$ do
    Let $\mathcal{S}$ be the skeleton of $v$;
    if at least three edges in $\mathcal{S}$ are blue but not permeable then
        result = “false”;
```

Next, we construct an algorithm that marks edges in the skeletons with a new attribute that can have three different values: `left`, `right` and `nil`. If the virtual edge (the edge whose twin edge is in the parent of v) in a skeleton of node v is marked `left`, then the pertinent graph of v must be embedded in such a way that there is a blue vertex on the boundary of the face left of the virtual edge. If the edge is marked `right`, a blue vertex must be on the boundary of the face right of the virtual edge. If the virtual edge is marked `nil`, there is no restriction on the embedding of the pertinent graph of v.

For each node v in the SPQR-tree, where the embedding of the parent node has already been fixed, we perform two steps:

1. We determine an embedding using the attribute of the virtual edge and the colors and attributes of the other edges.

2. We determine the attribute for the virtual edge in the skeleton of each child of v.

Only the skeletons of R- and P-nodes have more than one embedding, so the first step is only important for these node types. First we consider the case where v is an R-node. In this case, its skeleton has two embeddings. If the attribute on the virtual edge is `nil`, we can choose any of the two embeddings. Otherwise, we choose an embedding where there is a blue edge or vertex on the face left (right) of the virtual edge if the attribute was `left` (`right`). If none of the two embeddings has this property, no augmentation is possible. If v is a P-node, we can only choose among the admissible embeddings of the skeleton. Again, we choose an embedding according to the attribute and if no suitable embedding exists, there can be no augmentation.

Now we have to determine the attribute for the virtual edge of each child of v. Let $\mathcal{S}$ be the skeleton of v. For all edges in $\mathcal{S}$ that are either black or permeable, we pass `nil` to the corresponding child. Let L be the set of edges in $\mathcal{S}$ that are blue but not permeable.

First we consider the case that v is an S-node. If the attribute of the virtual edge is `nil`, we pass `nil` to every child that corresponds to an edge in L. Otherwise, the attribute on the virtual edge designates one of the two faces of the S-node skeleton as the one that must have a blue vertex on the boundary. For each edge e in L, we pass `left` to the corresponding child if this face is right of e and `right` otherwise.

To make the following descriptions more concise, we call a face *preferred*, if it has a permeable edge or a blue vertex on its boundary. If v is a P-node, L contains at most two edges. For each of edge e in L, there are three possible cases:

1. Exactly one of the faces with e on its boundary contains a blue edge. If this is the face on the right of e, we pass `left` to the child corresponding to e and `right` otherwise.

2. One of the faces with e on its boundary is preferred and the other is not. If the preferred face is left of e, we pass `right` to the child corresponding to e and `left` otherwise.

3. The faces left and right of e are both preferred or both contain only black edges and vertices. In this case, we pass `nil` to the child.

If v is an R-node, we also have to consider the faces left and right of each edge e in L. The same cases as for P-nodes apply, but there is one additional case that cannot occur in a P-node: Both faces left and right of e are not preferred but both contain a blue edge except e (if this happens in a P-node, there can be no augmentation). In this case, we pass `nil` to the corresponding child.

To start the process, we choose an arbitrary P- or R-node as the root of the SPQR-tree. If we choose an arbitrarily R-node, we select one of the two embeddings of the skeleton. If we select a P-node, we choose an arbitrary admissible embedding. Now we can compute the attributes we pass to the children of the node as stated above and compute an embedding for each skeleton of the SPQR-tree by applying the algorithm in depth first or breadth first sequence to all inner nodes of the tree.

This algorithm defines an embedding for each R- and P-node in the SPQR-tree and thus for the graph G. Since we touch each skeleton only once and the operations we perform for each skeleton can be done in time linear in the size of the skeleton, the embedding can be computed in linear time. Then we apply the algorithm from Section 6.1 to the fixed embedding. This algorithm either computes the list of edges that constitutes the planar connectivity augmentation or it signals that no augmentation is possible for this embedding.

Theorem 6.2. *Let Π be the embedding of G determined by the algorithm described above. G has a planar connectivity augmentation with respect to W if and only if there exists a planar connectivity augmentation for G with respect to the embedding Π.*

We will give an intuition why the theorem holds. It is obvious that a permeable skeleton of the SPQR-tree can be embedded arbitrarily. The same fact holds for black skeletons. We have to consider the blue skeletons that are not permeable. The embeddings that do not have a planar connectivity embedding have to be avoided. Hence if there are more than two blue vertices then every blue vertex has to be on a boundary of a face that contains another blue vertex. As a result, we have to avoid embeddings that have blue vertices where faces they belong to have only black vertices except the blue one if and only if an embedding exists that allows a planar connectivity augmentation. As we regard this fact and the embeddings of the skeletons of the S-, P- and R-nodes (Q-nodes are not needed to be regarded) we conclude the following:

S-node: If there are blue edges in the skeleton $\mathcal{S}$ of the S-node we have to embed the expansion graphs of the blue edges so that there is at least one blue vertex of each expansion graph in exactly one face the virtual edge belong to. These expansion graphs are expansion graphs of the children of the S-node, so we have to pass the same attribute to them, this means either left for the one face or right for the other face.

P-node: As we have at the most two blue edges in the skeleton of the P-node and chosen an admissible fix embedding we have to embed the expansion graph of the blue edges in the manner that there is at least one blue vertex in the same face as the permeable edge that represents a permeable expansion graph. Hence we pass the attribute right to the child of the P-node that represents the expansion graph of a blue edge if the permeable edge is ordered before the blue edge against clock-order and left otherwise.

R-node: The skeleton of an R-node has two embeddings in respect to the virtual edge if we ignore the virtual edge. We can treat the embeddings of the R-node as fix because the boundary of the faces are the same. Therefore we just need to take a look at the blue edges. We assume that there is a blue edge e and the faces that contain e have the following properties:

- one has additionally a permeable edge or a blue vertex on the boundary and
- the other has additionally a blue but not permeable edge on the boundary.

We have to embed the expansion graph of e so that there is at least one blue vertex on the face that contains another blue vertex or another blue vertex of the expansion graph of the permeable edge. So we introduce the attribute preferred to faces of skeletons that contain blue vertices or permeable edges. If both faces have permeable edges or a blue vertex the embedding of the expansion graph of the blue edge can be done arbitrarily. The same happens if both faces contain additional only blue but not permeable edges or black edges or black vertices. In that ranking we pass the corresponding attributes to the children.

As we traverse the rooted SPQR-tree top down and fix the embedding of every skeleton on the way we obtain a fixed embedding that contains all fixed embeddings of the skeletons of the SPQR-tree as a property of the SPQR-tree. Therefore this embedding contains the information whether a planar connectivity augmentation is possible. If one exists we have constructed one, if not, then there would not be a possibility to construct a feasible one. This can be tested easily with the algorithm for the fixed embedding.

The proof uses structural induction over the SPQR-tree. We first show that the claim holds for graphs whose SPQR-tree has only one inner node and then use induction to show that it holds for graphs whose SPQR-tree has more than one inner node.

Lemma 6.2. *G has a planar connectivity augmentation if and only if there is an embedding Π of G with a sequence of faces $f_1, \ldots, f_k$ with the following property:*

(**) for all $1 \leq i < k$, there is at least one vertex of W on the boundary between f_i and f_{i+1} and the boundaries of the faces f_i ($1 \leq i \leq k$) contain all vertices of W.

Proof. $\Leftarrow$ If there is an embedding Π of G with a sequence of faces $f_1, \ldots, f_k$ with property $(**)$ then we can create a dummy vertex d for each face f of the sequence and connect d with the blue vertices on the boundary of f. It is obvious that the graph constructed of the dummy vertices and its incident edges is connected because of property $(**)$. For each dummy vertex d we insert $\delta(d)+1$ edges: Between two blue vertices that are the end vertices of two adjacent edges of d we introduce an edge. We delete the dummy vertices d. It is obvious that the graph G_W induced of the vertices of W is connected and graph G is still planar. Therefore G has a planar connectivity augmentation.

$\Rightarrow$ If G has a planar connectivity augmentation, then there exists a minimum spanning tree of the subgraph G_W and an embedding Π that allows a planar connectivity augmentation. We root the minimum spanning tree at a vertex with degree 1 and we direct the edges in breath search to the leaves. Then we traverse the minimum spanning tree in preoder:

- For every original edge of G we introduce the face on the left to the list `facelist` and
- for every edge of F we introduce the original face of G that has to be split into the list `facelist`.

It is obvious that the embedding Π together with the sequence of faces of the list `facelist` has the property $(**)$. □

Now we are able to prove Theorem 6.2:

Proof. We have shown in Lemma 6.2 that permeable expansion graphs that are represented by permeable edges can be embedded arbitrarily. It is obvious that expansion graphs of black edges can also be embedded arbitrarily. Hence we have to consider the blue edges. Note that we call edges blue if they are blue but not permeable.
We use induction over the number n of inner nodes of the SPQR-tree of the graph G.

n $= 0$**:** G consists of two edges and two vertices and the SPQR-tree consists of two Q-nodes and an edge connecting them. The skeleton of the Q-nodes has only one embedding and therefore G has only one embedding. Is is obvious that the algorithm computes an embedding that allows a planar connectivity augmentation if one exists (firstly, the two vertices in G are connected and there is no need to augment the graph; therefore the minimum cardinality edge set is empty in all cases, and secondly, if at least one vertex of G is blue then the Q-node is permeable and by definition there exists an embedding Π and a sequence of faces so that property $(**)$ holds).

n $= 1$**:** The SPQR-tree of G consists of an inner node v' adjacent to Q-nodes. Let S' be the skeleton of v'. We distinguish three cases defined by the type of v'.

1. **S**-node: It is obvious that the expansion graphs of the edges of S' have only one embedding and are permeable if W is not empty. Since all vertices of the Q-nodes are contained in S', S' cannot have blue edges. Therefore if W is not the empty set it is obvious that S' is permeable and that S' is equal to G. Hence G can be embedded arbitrarily. Our algorithm passes attribute `nil` and works correctly.

2. **P**-node: It is obvious that the expansion graphs of the edges of S' have only one embedding and are permeable if W is not empty. Since all vertices of the Q-nodes are contained in S', S' cannot have blue edges. Since S' has only two vertices that are pole vertices, S' is permeable if W is not the empty set. Furthermore, this two vertices are connected so there is no need to augment the graph if W is not empty. Therefore G can be embedded arbitrarily. Our algorithm passes attribute `nil` and works correctly.

3. **R**-node: It is obvious that the expansion graphs of the edges of S' have only one embedding and are permeable if W is not empty. Since all vertices of the Q-nodes are contained in S', S' cannot have blue edges. Furthermore, it is obvious that S' is equal to G. S' has two embeddings but the boundaries of the faces do not change. Therefore if G has a planar connectivity augmentation both embeddings will have a sequence of faces so that property $(**)$ holds. Our algorithm passes attribute `nil` and works correctly.

$\mathbf{n} > 1$**:** Let v' be the node attached to v that contains the twin edge of e and S' the skeleton of v'. By induction the expansion graphs of the edges of S' are embedded so that the embedding allows a planar connectivity augmentation if one exists. We distinguish three cases defined by the type of v'.

1. **S**-node: Our algorithm gives the attribute `left` or `right` (according to the same face) for each blue edge in the skeleton of S' if S' has no permeable edges or blue vertices, `left` or `right` if S' has permeable edges or blue vertices. If S' has a blue vertex or a permeable edge it follows by Lemma 6.2 that there exists an embedding Π together with a sequence of faces that satisfy $(*)$. Furthermore, the coloring algorithm has marked the edges of S' so that S' represents the whole graph G. By induction, we know that there is an embedding Π' for each expansion graph of the blue edges together with a sequence of faces so that property $(**)$ holds if there is a planar connectivity augmentation. We have to distinguish two cases.

   (a) If G has a planar connectivity augmentation then there exists an embedding Π of G together with a sequence of faces such that property $(**)$ holds. As S' represents the whole graph, then there have to be such a sequence of faces. Hence the permeable edge or blue vertex are on the boundary of faces f_1 and f_2, both faces can but at least one must be contained in this sequence but at least one must be. To obtain a sequence of faces with property $(**)$

the expansion graph of a blue edge has to be embedded so that there is at least one blue vertex on the boundary of face f_1 or face f_2 and there is a sequence of faces of the expansion graphs so that $(**)$ holds. By induction this embedding is constructed for the expansion graph of each blue edge. Hence we can concatenate the sequence of faces according to the embeddings Π' of the expansion graphs in S'. As our algorithm passes either `left` or `right` to the blue edges we get an embedding Π with a planar connectivity augmentation.

(b) If G has no planar connectivity augmentation then there does not exist an embedding Π of G together with a sequence of faces such that property $(**)$ holds. Since S' represents the whole graph, there is no such sequence of faces. Therefore the embedding we get by the embedding algorithm cannot allow a planar connectivity augmentation.

If S' does not have permeable edges or blue vertices but blue and black edges, we know by induction that the expansion graphs have an embedding Π that allows a planar connectivity augmentation if one exists. As they are not permeable and there have to be an embedding Π' together with a sequence of faces so that property $(**)$ holds, we have to concatenate the sequence of faces in S' either according to face f_1 or to face f_2. It is obvious that this sequence of faces satisfies property $(**)$ if a planar connectivity augmentation exists. Our algorithm passes either `left` or `right` for this case such that it allows a planar connectivity augmentation if one exists.

2. **P**-node: Our algorithm computes an admissible embedding. By Lemma 6.2 we know that for every expansion graph of a permeable edge there exists an embedding Π together with a sequence of faces so that property $(*)$ holds. It is obvious that the algorithm cannot compute an embedding that allows a planar connectivity augmentation if none exists. We assume that G has a planar connectivity augmentation. By induction there is a planar connectivity augmentation in each expansion graph of a blue edge e'' in S'. To concatenate the sequence of faces so that property $(**)$ holds we have to embed the expansion graphs of the blue edges so that there is a blue vertex in the boundary of the face between its blue edge and the adjacent permeable edge. If S' contains no permeable edge and two blue edges, we have to embed the expansion graphs of the two blue edges so that the face bounded by the two blue edges in S' contains blue vertices of both expansion graphs on the boundary in the embedding Π. Such an embedding is constructed by induction. We concatenate the sequence of faces according to the embedding of the expansion graphs of S' and get an embedding that allows a planar connectivity augmentation. For each blue edge in S' the algorithm passes following attributes:

(a) Exactly one of the faces with e on its boundary contains a blue edge. If this is the face on the right of e, we pass `left` to the child corresponding to e and `right` otherwise.

(b) One of the faces with e on its boundary is preferred and the other not. If the preferred face is left of e, we pass `right` to the child corresponding to e and `left` otherwise.

(c) The faces left and right of e are both preferred or both contain only black edges and vertices. In this case, we pass `nil` to the child.

Hence our algorithm works correctly.

3. **R**-node: By Lemma 6.2 we know that for every expansion graph of a permeable edge there exists an embedding Π together with a sequence of faces so that property $(*)$ holds. By induction there is a planar connectivity augmentation in each expansion graph of a blue edge if one exists. Since the skeleton of the R-node has only two embeddings we can concatenate the faces according to the embeddings of the expansion graph in S'. If this sequence of faces satisfies $(**)$ then there exists a planar connectivity augmentation. Therefore our algorithm works correctly.

□

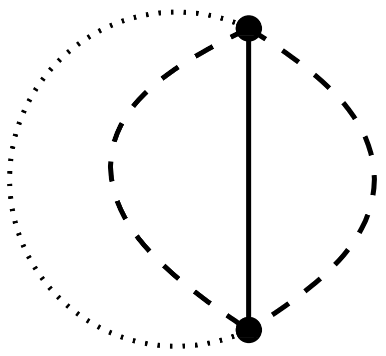


Figure 6.7: Sorting the edges of a blue P-node: blue edges are represented by dashed lines and the permeable edge by a dotted line.

For a pseudo-code description see Algorithm 11.

We can state the following theorem.

Theorem 6.3. *Given a planar biconnected graph $G = (V, E)$ and a subset of vertices $W \subseteq V$. The algorithm* `BiconnectedAugmenter` *tests correctly whether a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set. It runs in time $O(|V|)$.*

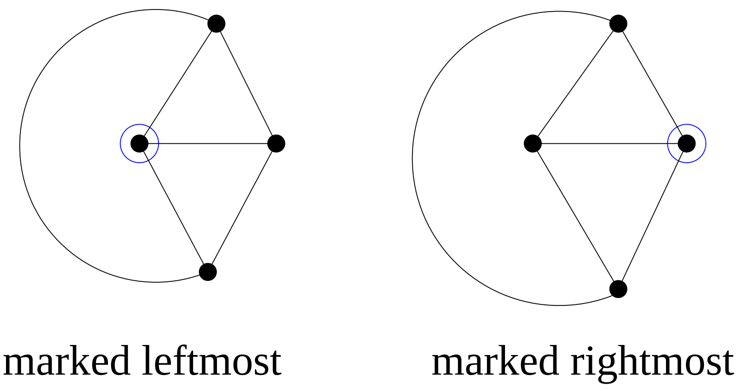


Figure 6.8: Swapping the leftmost or rightmost marked nodes.

Algorithm 10: Algorithm `CalculateEmbedding` determines a feasible embedding of the graph, if there is one and none, if there is none.

```
Input: A biconnected planar graph $G$ and its $SPQR$-tree.
Result: An embedding of graph $G$.
forall the $P$-nodes which are blue do
    Sort and mark the first blue edge leftmost and the second rightmost;
    Sort all permeable edges leftmost;
forall the $R$-nodes which are blue and not permeable do
    Extend algorithm findpath to mark the $R$-node with the labels $s$, $t$ or $nil$;
forall the $R$-nodes which are blue and not permeable do
    forall the faces $f$ in the skeleton of the $R$-node which contain at least two of
    the following: blue nodes, blue edges and permeable edges do
        Traverse the boundary of $f$ and if there is a blue edge representing an $R$-node
        $R$, swap the skeleton of $R$ if $R$ is marked $t$;
    if father which is not an $R$-node is marked leftmost then
        Swap the $R$-node if it is marked $t$;
    if father which is not an $R$-node is marked rightmost then
        Swap the $R$-node if it is marked $s$;
Construct an embedding from the $SPQR$-tree;
```

Algorithm 11: The algorithm `BiconnectedAugmenter` computes a planar connectivity augmentation for a planar biconnected graph G and a subset W of the vertices, if it exists.

```
Input: A biconnected planar graph $G$ and a subset $W$ of its vertices,
Result: true if and only if there is a planar augmentation for $W$; in the positive
        case an embedding $\Pi$ and a minimum cardinality augmenting edge set will
        be computed.
Calculate the SPQR-tree $\mathcal{T}$ of $G$;
Make an arbitrary node $r$ which is not a $Q$-node the root of $\mathcal{T}$;
MarkEdgesPhase1($r, W$);
MarkEdgesPhase2($r, W$);
BiconnectivityFeasibilityCheck($\mathcal{T}$);
Embedding $\Pi$ = CalculateEmbedding;
return FixedEmbeddingAugmenter($\Pi, W$);
```

6.3 The Algorithm for the Connected Case

In the last section we have dealt with the problem of finding an induced subgraph of W in a biconnected graph. Hence we already know how to deal with the biconnected blocks of the graph G using the SPQR-tree. To solve the connected case, we first build the BC-tree of G. This tree has two types of nodes: The c-nodes correspond to cut-vertices of G and the b-nodes to biconnected components (blocks). There is an edge connecting a c-node and a b-node, if the cut-vertex is contained in the block corresponding to the b-node. Taking this structure into account we can solve the problem also for connected graphs in linear time.

We split the connected graph G into biconnected components using the BC-tree and we split the biconnected blocks into triconnected components using the $SPQR$-tree. In the last section we have dealt with the problem of finding an induced subgraph of W in a biconnected graph. Hence we already know how to deal with the biconnected blocks of the graph G using the $SPQR$-tree. The blocks which are connected to each other by a cut vertex can be embedded in each face they belong to. Therefore every blue block has to be embedded in the same face because we will have to connect them if a feasible planar connectivity augmentation exists. Further, it is obvious that we have to operate only on the smallest subtree bc of the BC-tree containing all blue blocks. Such a subgraph induced of bc can look like the graph G in figure 6.9. If we connect the cut vertices of the blocks of bc, which are not blue and if the result of the planarity test is positive (that means that the blocks can be embedded in the same face), we will be able to connect the blue blocks. It is obvious that at least one cut vertex is also connected to a blue block. If the planarity testing results true, the blue blocks will be able to be connected in the same face (see figure 6.9). Otherwise we have the same situation as shown in figure 6.10. The blue blocks cannot be connected in this case because the cut vertices do not lie on the same face. In the positive case, we mark all cut vertices that are not blue red. For each blue biconnected component its red vertices play a special role since they represents the connection to the other blue biconnected component in *bc*. Consequently, its blue vertices has to be connected with the red vertices to ensure a planar connectivity augmentation for all blue vertices represented by the vertex subset W. Hence, for each biconnected component that is marked blue we apply the Algorithm 11 of the biconnected case presented in the last section on its blue and red vertices. In the coloring algorithm, Algorithm 7, we differ between blue and red vertices: clearly, the red vertices has to appear in the beginning or the end of the sequences. Consequently, we assign the attribute `blue` to a virtual edge e if e has no attribute `permeable` created by the blue vertices of its expansion graph but its expansion graph contains a red vertex.

Corollary 6.1. *A connected series-parallel planar graph G with one not connected subgraph G_W induced by the vertex subset $W \subseteq V$ can be augmented if and only if*

- *all P-nodes of its $SPQR$-tree contain at the most two blue and not permeable edges and*
- *the red vertices of the graph G can be connected with a dummy vertex so that the graph keeps its planarity.*

Corollary 6.2. *A connected planar graph G with one not connected subgraph G_W induced by the vertex subset $W \subseteq V$ can be augmented if and only if*

- *all P-nodes of its $SPQR$-tree contain no blue pole vertex and at the most two blue but not permeable edges,*
- *in every R-node there is no cycle of black vertices and black edges which separate blue vertices and blue edges and*
- *the red vertices of the graph G can be connected with a dummy vertex so that the graph keeps its planarity.*

6.4 The Algorithm for the General Case

For general graphs we apply the Algorithm 24 to each connected component. Then we transform the connected components so that they have at least one blue and all red vertices on the outer face. Finally we connect the blue vertices on the outer face.

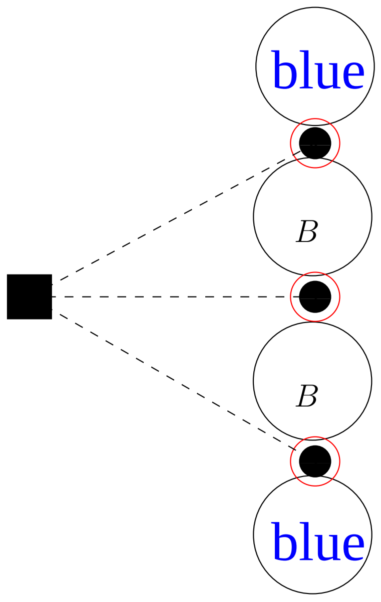


Figure 6.9: The graph G induced of the subtree bc and the test if the augmentation is not feasible.

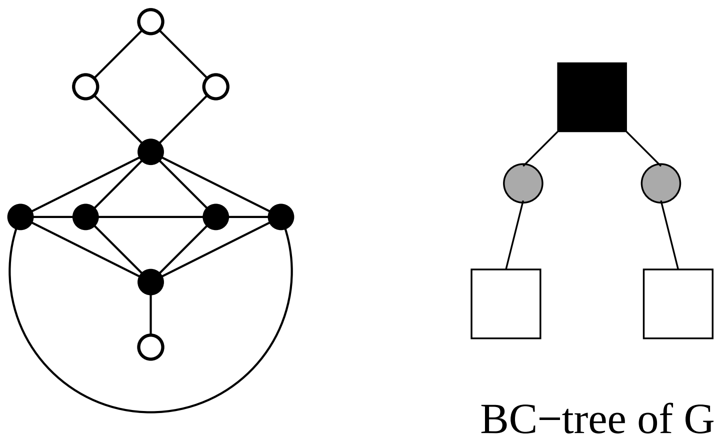


Figure 6.10: No feasible way to connect the blue vertices (drawn as hollow circles) and to keep planarity of graph G.

Algorithm 12: Algorithm `ExtendedCheckAugmentability` checks if a planar connectivity augmentation for W in a connected graph G exists.

```
Input: A connected planar graph.
Result: An embedding of $G'$ if $G$ can be augmented, otherwise "false".
Calculate $BC$-tree of graph $G$;
forall the blocks $B$ of $BC$-tree do
    if $B$ contain a blue vertex then
        Mark $B$ blue;
Calculate the smallest $BC$-subtree $bc$ containing all blue blocks;
forall the $C$-nodes $c$ in $bc$ do
    if $C$-node $c$ is not blue then
        mark $c$ red;
forall the blocks $B$ of $bc$ which are not blue do
    Join all red nodes of the blocks $B$ with a single dummy vertex;
    Test planarity;
forall the blocks which are blue do
    Apply changed algorithm CheckAugmentability with red nodes;
```

Algorithm 13: Algorithm `GeneralCheckAugmentability` checks if a planar connectivity augmentation for W in a general graph G exists.

```
Input: A general graph $G$.
Result: An embedding of $G'$ of $G$ can be augmented, otherwise "false".
forall the connected components $C_i$ of the graph $G$ do
    ExtendedCheckAugmentability;
Sort all $C_i$ in clockwise order and traverse them in this order;
forall the $c_i$ which contains at least one blue or red vertex do
    Choose a face $f$ with at least one blue or red vertex in the boundary as outer
    face;
    Save the blue or red vertex in a list listvertex;
forall the vertices in listvertex do
    Connect them by inserting clockwise a path that is a cycle minus one edge;
```

Chapter 7

c-Planarity Testing

7.1 c-Connected Clustered Graphs

In this chapter we consider the c-planarity of a c-connected clustered graph $C = (G, T)$. There exist two planarity testing algorithms for this class of clustered graphs by Feng in [29, 30] that runs in quadratic time and by Dahlhaus in [18, 20] who gives a linear running time improvement.

In this chapter we develop an additional linear time algorithm for this problem. Our aim is more to create an other view-point on the problem than to show a new algorithmic result since an elegant algorithm has already been developed by Dahlhaus in [18, 20] first as mentioned before. In this chapter we focus on a c-planarity characterization with forbidden minors that surprisingly can be achieved for the c-connected planar clustered graph. By this, we are able to understand the root of the problem. Finally, using this characterization, we show an additional algorithmic realization of the problem deciding c-planarity and in the positive case achieving a c-planar embedding.

Theorem 7.1. *Let a c-connected clustered graph $C = (G, T)$ be given and let G be planar. C is c-planar if and only if for each cluster ν of T C has no minor isomorphic to the clustered graphs of Figures 7.1 and 7.2.*

Proof. We know that a completely connected clustered graph is c-planar if and only if it is planar (see [15, 54] and Chapter 8). Furthermore, it is a well-known fact that a c-connected clustered graph with an underlying graph isomorphic to a tree is c-planar [21].

Therefore, given a c-connected planar clustered graph, it is not c-planar if and only if it has no planar completely connected clustered super-graph. In this case there has to exist a cluster ν such that $G(\nu)$ isolates $G - G(\nu)$ in each planar embedding of G.

Since G is planar, G has neither a $K_{3,3}$ nor a K_5 minor. Since C is not c-planar we may assume that augmentation to a completely connected clustered graph yields a $K_{3,3}$ or K_5, respectively.

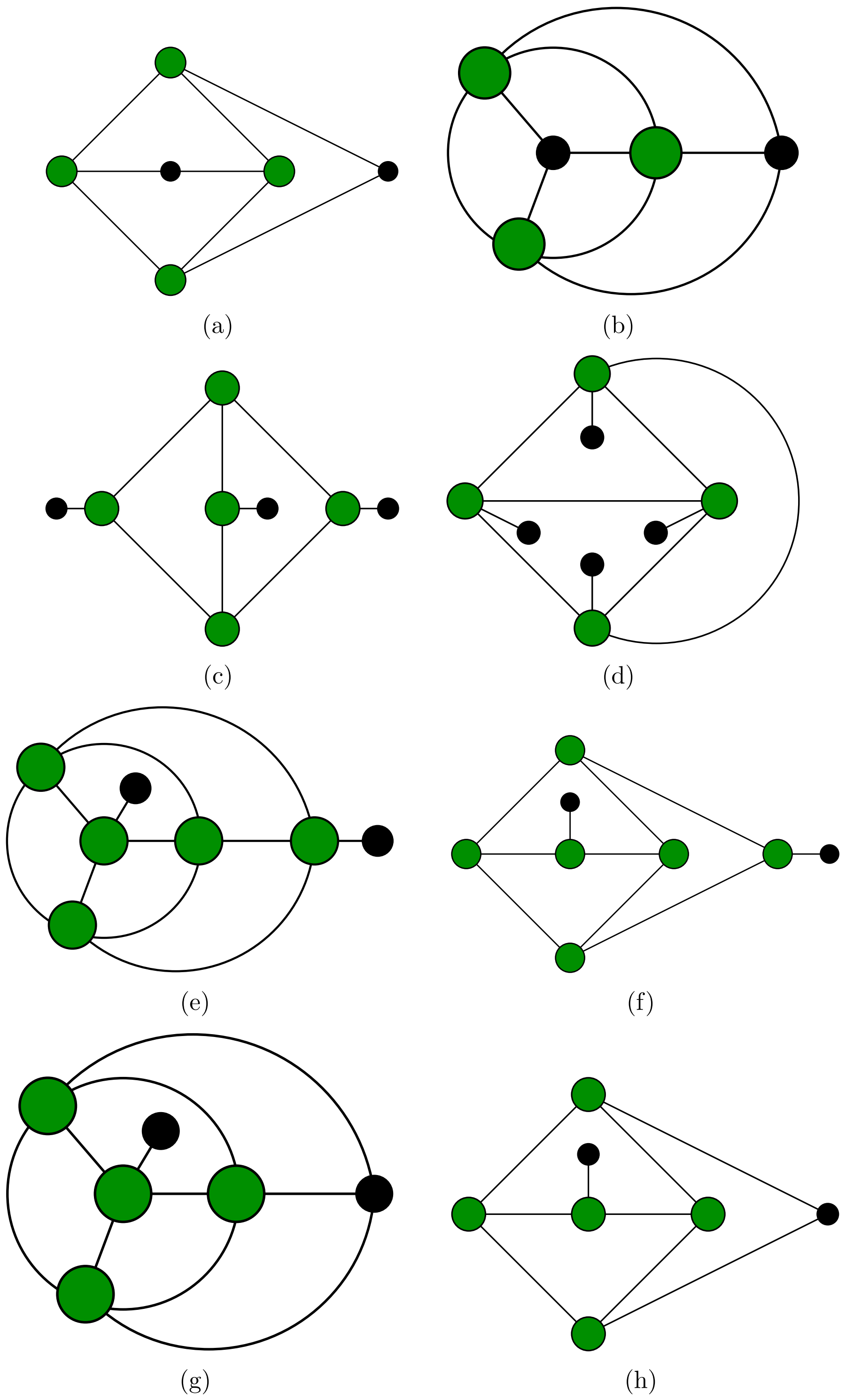


Figure 7.1: Forbidden clustered minors of a c-connected clustered graph for a cluster ν (green vertices are belonging to $G(\nu)$, black vertices to $G - G(\nu)$)

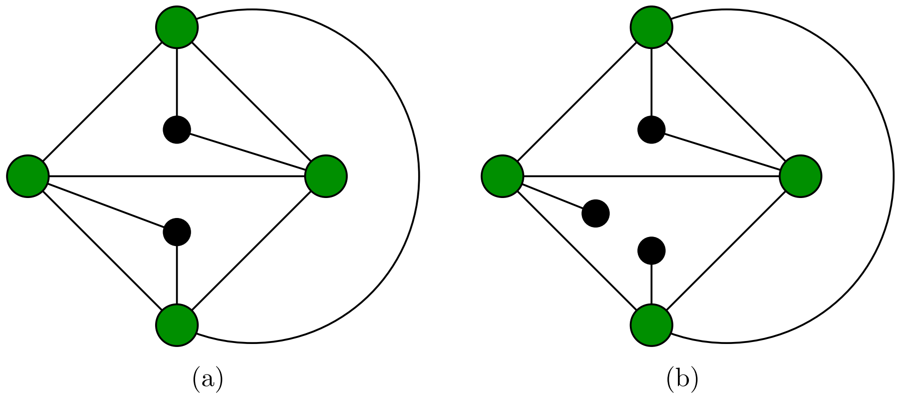


Figure 7.2: Forbidden clustered minors of a c-connected clustered graph for a cluster ν (green vertices are belonging to $G(\nu)$, black vertices to $G - G(\nu)$)

Let $V_{G(\nu)}$ be the vertex set of $G(\nu)$ and $V_{G-G(\nu)}$ the vertex set of $G - G(\nu)$. Furthermore, let $n^* = |V_{G(\nu)}|$.

Therefore, we may suppose that we have the following cases for a c-connected clustered graph that is not c-planar:

1. $K_{3,3}$:

   (a) $n^* =$ 1,2,3: $G(\nu)$ is a tree since $K_{3,3}$ has cycles with length at least 4.

   (b) $n^* = 4$: either $G(\nu)$ is a tree (by choosing three vertices adjacent to an other one) or $G(\nu)$ is equal to C_4, the cycle of length 4. In the second case there is a unique edge e between the two vertices of $G - G(\nu)$ after choosing the four vertices of $G(\nu)$. This constellation is visualized in Figure 7.1(a).

   (c) $n^* = 5$: $G(\nu)$ is a $K_{2,3}$.

   The assumption that there is a vertex connected to $K_{2,3}$ introducing a $K_{3,3}$ leads to a contradiction that G is planar. Therefore, we have more than one vertex belonging to $G - G(\nu)$. Let V_3 be the partition of $K_{2,3}$ with three vertices.

      i. Let us suppose that we have two vertices. Then one vertex of $G - G(\nu)$ is connected to two vertices of V_3 and the other vertex is connected to the remaining vertex of V_3. This constellation is visualized in Figure 7.1(h).

      ii. Let us suppose that we have three vertices. Then each vertex is connected to a different vertex of V_3. This constellation is visualized in Figure 7.1(c).

2. K_5:

   (a) $n^* =$ 1,2: $G(\nu)$ is a tree since K_5 has cycles with length at least 3. The existence of K_5 contradicts the assumption that G is planar.

(b) $n^* = 3$: $G(\nu)$ is equal to C_3, the cycle of length 3. Then there is a unique edge e between the two vertices of $G - G(\nu)$ after choosing the three vertices of $G(\nu)$. This constellation is visualized in Figure 7.1(b).

(c) $n^* = 4$: $G(\nu)$ is equal to K_4.

The assumption that there is a vertex connected to all vertices of K_4 introducing a K_5 leads to a contradiction that G is planar. Therefore, we have more than one vertex belonging to $G - G(\nu)$. Let V_4 be the vertex set of $G(\nu)$.

i. Let us suppose that we have two vertices in $G - G(\nu)$. Then

A. one vertex of $G - G(\nu)$ is connected to three vertices of V_4 and the other vertex is connected to the remaining vertex of V_4. This constellation is visualized in Figure 7.1(g), or

B. one vertex is connected to two vertices of V_4 and the other to the remaining vertices of V_4. This constellation is visualized in Figure 7.2(a).

ii. Let us suppose that we have three vertices in $G - G(\nu)$. Then one vertex is connected to two vertices of V_4, and the other two are connected each to a different vertex of V_4. This constellation is visualized in Figure 7.2(b).

iii. Let us suppose that we have four vertices in $G - G(\nu)$. Then each vertex is connected to a different vertex in V_4. This constellation is visualized in Figure 7.1(b).

□

For example, on the left-hand side of Figure 7.3 the clustered graph has a clustered minor isomorphic to the clustered graph of Figure 7.1(b), as visualized on the right-hand side of Figure 7.3.

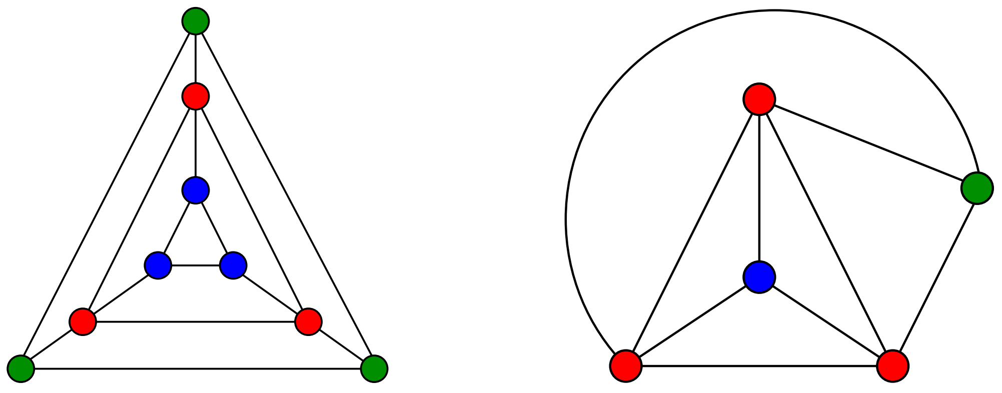


Figure 7.3: A c-connected clustered graph on the left-hand side, vertices of a cluster are visualized red, blue and green, respectively. The clustered graph on the left-hand side is isomorphic to a forbidden minor.

In Figure 7.1(a) and 7.1(b), notice that merging the vertices of $G - G(\nu)$ yields a planar graph. On the other hand, merging the vertices of $G - G(\nu)$ of the other minors cause non-planarity.

In Figures 7.1 and 7.2, notice that each of the clustered graphs has a triconnected minor except the graph of Figure 7.1(c). Analyzing the corresponding triconnected minors we observe that we can classify them (see Figure 7.4).

We may conclude a special characterization.

Corollary 7.1. *Let $C = (G, T)$ be a c-connected clustered graph and G be outer-planar. Then C is c-planar.*

Proof. Recall that an outer-planar graph is a series-parallel graph with no minors isomorphic to K_4 and $K_{2,3}$ and is planar. The statement follows immediately by Theorem 7.1 since the forbidden minors cannot be included by definition. □

Using the previous results and Theorem 2.3 we can state the following corollary.

Corollary 7.2. *Let $C = (G, T)$ be a c-connected clustered graph. C is c-planar if and only if there is no isolating cut cluster in C.*

Proof. Follows immediately by Theorems 2.3 and 7.1. □

Given a *c*-connected planar clustered graph, our strategy is to create a planar embedding such that for all clusters ν of T all vertices of $G - G(\nu)$ are outside of $G(\nu)$ if and only if such an embedding exists. Hence, if the created planar embedding isolates $G - G(\nu)$ from $G(\nu)$ then C is not *c*-planar.

Notice that *c*-connectivity of a clustered graph can be tested easily in linear time by recursively (bottom up level by level in the clustered tree) shrinking the connected vertices of a cluster. A cluster is not connected if the shrinked cluster contains more than one vertex.

7.1.1 The Algorithm for the Triconnected Case

Let $C = (G, T)$ be a *c*-connected planar clustered graph and G triconnected.

Choose a planar embedding of G such that there is an edge only belonging to the root cluster on the boundary of the outer face. Since we have at least one cluster other than the root cluster such an edge must exists.

Hence G is triconnected we have a unique planar embedding Γ beside mirroring. Therefore, it is easy to see that we can determine isolations of $G - G(\nu)$ from $G(\nu)$ in Γ simply by shrinking each cluster ν bottom up level by level in the cluster tree. Obviously, a created self-loop out of $G(\nu)$ indicates an isolation of $G - G(\nu)$.

This immediately leads us to Algorithm 14 for the triconnected case.

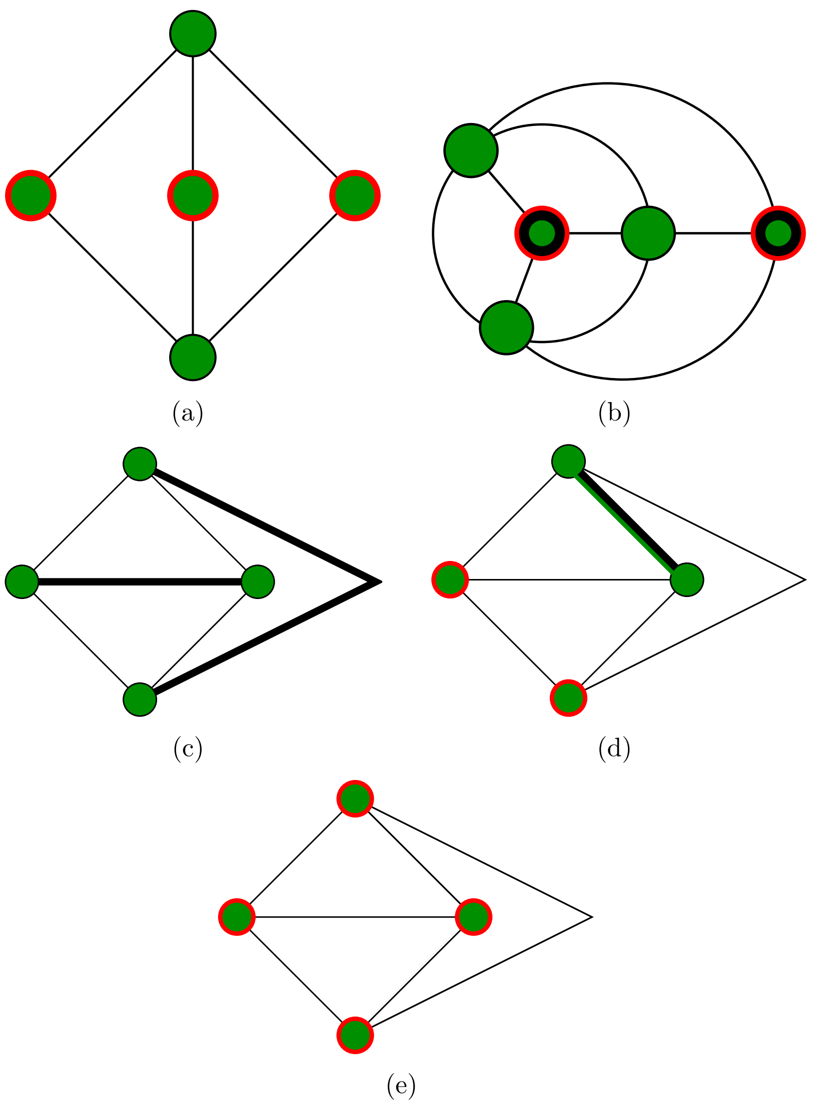


Figure 7.4: Characterization of the forbidden clustered minors of Figures 7.1 and 7.2; green vertices are belonging to $G(\nu)$, red circled vertices are green vertices that correspond to cut vertices connected to connected components containing vertices of $G - G(\nu)$, red-black circled vertices can be seen as red circled vertices or vertices of $G - G(\nu)$, bold edges are virtual edges that contain vertices of $G - G(\nu)$ if black, or a path between green vertices if green

Algorithm 14: Algorithm `Test_cplanar_triconnected` that tests c-planarity for a c-connected clustered graph $C = (G, T)$ with G planar and triconnected.

```
Input: A c-connected clustered graph $C = (G, T)$, $G$ planar and triconnected
Result: true, if and only if $C$ is c-planar, otherwise false.
Choose a planar embedding $\Gamma$ with an edge $e$ on the boundary of the outer face that
belongs to the root cluster;
forall the cluster $\nu$ bottom up level by level do
    Shrink $\nu$ in $\Gamma$ to a node;
    Treat modified embedding as $\Gamma$ from now on;
    if a self–loop is created then
        return false;
return true;
```

7.1.2 The Algorithm for the Biconnected Case

Let us assume that $C = (G, T)$ is c-connected planar and its underlying graph G is biconnected.

To investigate the structure of C, we first omit triconnected minors of G, and assume that G is series-parallel.

Theorem 7.2. *Let $C = (G, T)$ be a c-connected clustered graph and G series-parallel. C is c-planar if and only if it contains no minor isomorphic to the clustered graph of Figure 7.1(c).*

Proof. Since the other clustered minors visualized in Figure 7.1 and 7.2 have a triconnected minor their existence in a series-parallel graph leads to a contradiction. □

From now on we investigate in the c-planarity testing algorithm for the biconnected case. First, we build the SPQR-tree T of G. We make the following observations.

Theorem 7.3. *Let $C = (G, T)$ be a c-connected clustered graph and G series-parallel. Let $\mathcal{T}_i$ be the $SPQR-trees$ of the biconnected components C_i of G for all $i = 1, \ldots, n$, $n \in N$.*

C is c–planar if and only if for all clusters ν there are no isolations of $G - G(\nu)$, means ν is no isolating cut cluster, in the pertinent graphs of the P-nodes of all SPQR-trees of G.

To prove this theorem we prove first the following theorem.

Theorem 7.4. *Let $C = (G, T)$ be a c-connected clustered graph and G series-parallel and biconnected. Let $\mathcal{T}_i$ be the SPQR-trees of the biconnected components of G. Let $\mathcal{C}_\nu := \{\mathcal{V} \in 2^{V(\nu)} | \mathcal{V}_\nu$ is a circle in the expansion graph of a P-node and contains both pole vertices$\}$ be the sets of the vertex sets belonging to cluster ν of all circles in the expansion graphs of P-nodes that contain both pole vertices.*

If for every P-node of the SPQR-trees the following property $(*)$ holds, then C is c-planar:

($*$) *For every cluster ν in every P-node, there is at the most one circle $\mathcal{V}_\nu$ of $\mathcal{C}_\nu$ such that*

1. *the union of the expansion graphs of two children $\wp_1$ and $\wp_2$ contains $\mathcal{V}_\nu$ and*
2. *for $i \in \{1,2\}$ the cut of $G - G_\nu$ with the expansion graph of child $\wp_i$ is nonempty.*

Proof. Since C is c-connected, isolations of $G - G(\nu)$ for a cluster ν can only occur if the pertinent graph of a P-node isolates $G(\nu)$. The skeleton of an S-node S is a cycle, therefore has exactly two faces and one embedding. Therefore isolations can only happen in the pertinent graphs of the virtual edges of S (it can be shown that otherwise S must be an R-node). Isolations of $G(\nu)$ can only happen if the pertinent graph of a P-node contain at least three cycles $\mathcal{V}_\nu$ with the conditions above (see for an example Figure 9.7).

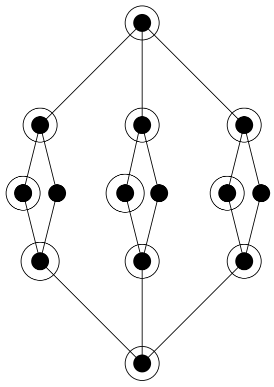


Figure 7.5: Isolations in the expansion graph of P-nodes: The circled vertices are vertices of a connected cluster ν, the others belong to $G - G(\nu)$

This proves the theorem. □

Theorem 7.3 follows from Theorem 7.4.

Theorem 7.5. *Let $C = (G,T)$ be a c-connected clustered graph and G series-parallel. Let $\mathcal{T}_i$ be the SPQR-trees of the biconnected components of G.*

C is c–planar if and only if there is no skeleton of a P-node P in the SPQR-trees containing at least three disjoint paths X from one pole vertex to the other of a cluster ν with the following properties:

Each of the pertinent graphs of the skeletons of the children of P containing X do contain vertices of $G - G(\nu)$.

Proof. Follows immediately from Theorems 7.1, 7.2 and 7.3. □

By Corollary 7.2, to test c-planarity of a c-connected planar graph with G biconnected we have to check isolations in the pertinent graphs of every P- and R-node of its SPQR-tree $\mathcal{T}$.

In Figure 7.6 a c-connected planar clustered graph $C = (G, T)$ is visualized with G biconnected. As we can see, this clustered graph is obviously not c-planar.

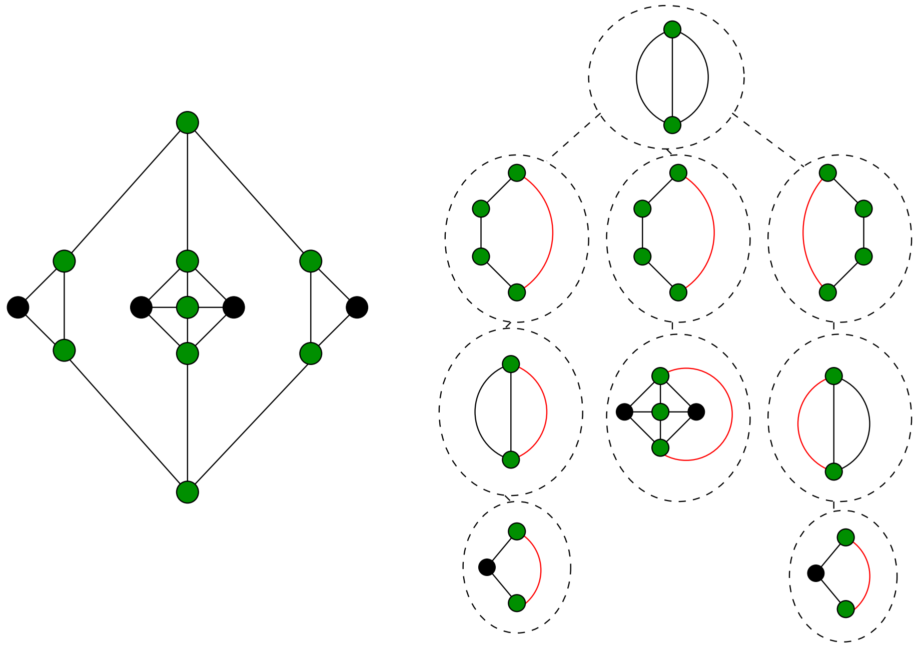


Figure 7.6: Example of a clustered graph (left-hand) and its SPQR-tree (right-hand) we use in this section to demonstrate the c-planarity testing algorithm; vertices of cluster ν are visualized green

Definition 7.1. *Let $C = (G, T)$ be a c-connected planar clustered graph, G biconnected and $\mathcal{T}$ the SPQR-tree of G. In a skeleton S of a node v of $\mathcal{T}$: A path between the pole vertices s, t of S that consists of all nodes and edges belonging to the lowest common ancestor of s, t in T is called an* lca(s,t)-path.

We observe that for two disjoint clusters ν and μ their corresponding ν- and μ-path are disjoint. On the other hand, if ν is a father of μ, the μ-path is contained in the ν-path.

First, we construct an auxiliary copy of T, say H_T. For each cluster ν of T, we determine the isolations of $G - G(\nu)$ from $G(\nu)$ using H_T:

For each skeleton of a child v of a P-node in T we test whether there is a lca(s,t)-path between its two pole vertices s, t. If there is one, then we mark the corresponding virtual edge in the P-node with the attribute *lca(s,t)-path*. Furthermore, if there is an other edge not belonging to a cluster of the subtree of lca(s,t), then we mark the same virtual edge with an additional attribute *lca(s,t)-outer*. Observe that if there are at least three virtual edges marked both lca(s,t)-path and lca(s,t)-outer in a P-node, then there is an isolation of $G - G(\text{lca}(s,t))$ from $G(\text{lca}(s,t))$, means lca(s,t) is an isolating cut cluster, in the corresponding pertinent graph and C cannot be c-planar.

To apply, if necessary, the previous mentioned attributes lca(s,t)-path and lca(s,t)-outer to each virtual edge in a skeleton of a node of T, we do some shrink operations in H_T.

First, we add the attribute $lca(u_1, u_2)$-path to each original edge $e = (u_1, u_2)$ of C in a skeleton of $\mathcal{T}$ and $\mathcal{H}_\mathcal{T}$ (see Figure 7.7). For further calculations we omit the Q-nodes in the SPQR-trees. We part the algorithm for marking the edges in two parts, first called `MarkEdgesI` and the second named `MarkEdgesII`.

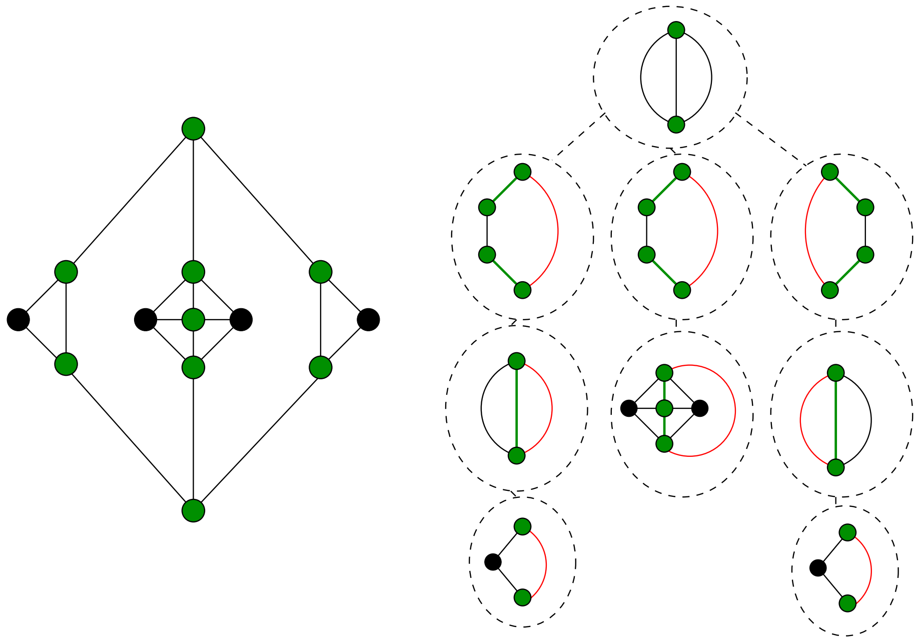


Figure 7.7: Continuation of Figure 7.6: After calling the preprocessing step; ν-paths visualized green

Traversing $\mathcal{H}_\mathcal{T}$ bottom up level by level, for each skeleton S of a node v of $\mathcal{H}_\mathcal{T}$ we add the attribute lca(s,t)-path to the twin edge of v in the skeleton of the father node if and only if there is a path between both pole vertices s, t in S using only edges that belong to the subtree of lca(s,t) in T (see Figure 7.8). We can test this property very easily by shrinking those edges: If in the resulting skeleton s is merged with t, we have such a path. Additionally, whether there is a cluster that does not belong to the subtree rooted by lca(s,t) in T can also be easily verified by level calculations in T. Since the clustered graph is c-connected there exists an edge that belongs to a cluster with depth smaller than the depth of lca(s,t) in T.

Furthermore, if there is an other edge e in S other than the reference edge and not marked lca(s,t)-path we add the attribute lca(s,t)-outer to e and to the twin edge. Notice that we also mark the corresponding edges in the skeletons of $\mathcal{T}$. For a detailed pseudo-code description of Algorithm `MarkEdgesI` see Algorithm 15.

Observe that we do not mark the reference edges of each skeleton of the SPQR-trees in

Algorithm 15: Part I: Algorithm `MarkEdgesI`.

```
Input: A $c$-connected planar clustered graph $C = (G, T)$, $G$ biconnected
Result: all edges other than reference edges are marked lca($s$,$t$)-path or lca($s$,$t$)-
        other
$\mathcal{T}$ = SPQR-tree of $G$;
$s$,$t$ = pole vertices;
forall the edges $e$ of $G$ do
    Add attribute lca(source($e$),target($e$))-path to $e$;
forall the nodes $v$ of $\mathcal{T}$ bottom up level by level do
    $S$ = Skeleton of $v$;
    if $v$ is an S- or R-node then
        Calculate lca($s$,$t$)-path in $S$;
        if lca($s$,$t$)-path exists then
            Add attribute lca($s$,$t$)-path to twin edge of $S$;
        if exists edge $e$ not of lca($s$,$t$)) in $S$ then
            Add attribute lca($s$,$t$)-other to $e$ and twin edge of $S$;
    if $v$ is a P-node then
        if one edge has attribute lca($s$,$t$)-path then
            Add attribute lca($s$,$t$)-path to twin edge of $S$;
        if one edge has attribute lca($s$,$t$)-other then
            Add attribute lca($s$,$t$)-other to twin edge of $S$;
```

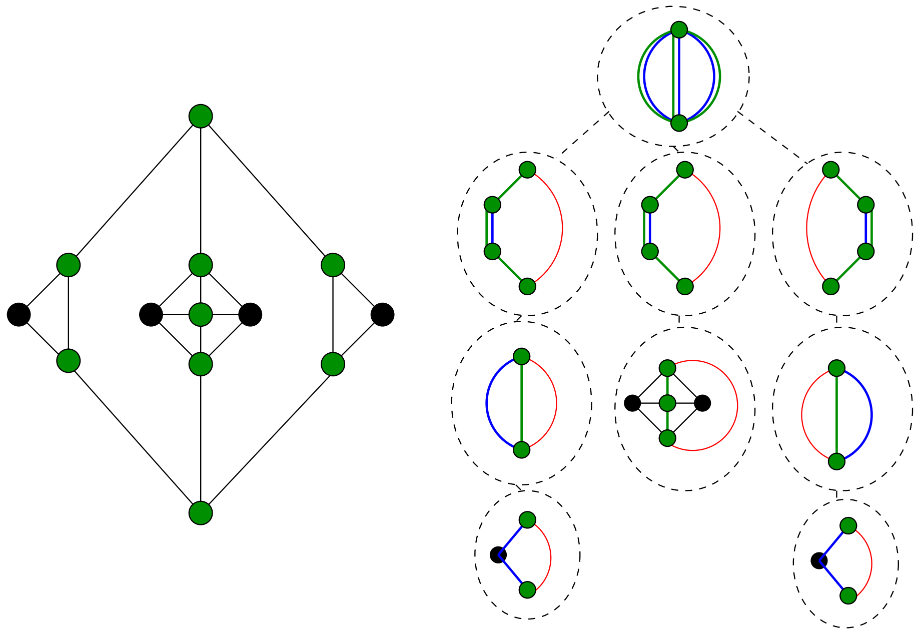


Figure 7.8: Continuation of Figure 7.6: After calling the algorithm `MarkEdgesI`; ν-paths visualized green, ν-others visualized blue

the first traversal. Next we mark them by traversing $\mathcal{H}_\mathcal{T}$ top down level by level (see Figure 7.9). In the skeleton of root node of $\mathcal{H}_\mathcal{T}$ all edges are marked. Therefore, we hand the attributes of a skeleton S of a node v down to the reference edges r of its child nodes by simply analyzing their corresponding edges e_r in the skeleton of v. Again, if those edges e_r contain an lca(s,t)-path, we give the attribute lca(s,t)-path to r. An analogous criteria hold for the attribute lca(s,t)-other. Notice that we also mark the corresponding edges in the skeletons of $\mathcal{T}$.

For a detailed pseudo-code description of Algorithm `MarkEdgesII` see Algorithm 16.

After finishing both traversals, it is easy to see that isolations (in form of an isolating cut cluster) are easy to verify: If a skeleton of a P-node in $\mathcal{T}$ contains more that two edges marked both lca(s,t)-path and lca(s,t)-outer, C is not c-planar. In our example clustered graph in Figure 7.9 the rooted P-node satisfy this property.

Furthermore, we apply the algorithm for the triconnected case of Section 7.1.1 to each skeleton of an R-node also taking the marked edges into account (but we do not shrink the reference edge). For example, in Figures 7.9 and 7.10 the R-node has this property because after the shrinking operation the vertex of the self-loop connects two blocks beside the reference edge. If the return value is `false`, C is not c-planar.

We call the algorithm `FeasibilityCheck` that tests those properties. For a detailed pseudo-code description of Algorithm `FeasibilityCheck` see Algorithm 17.

The two algorithms `MarkEdgesI` and `MarkEdgesII` can both be implemented in linear time

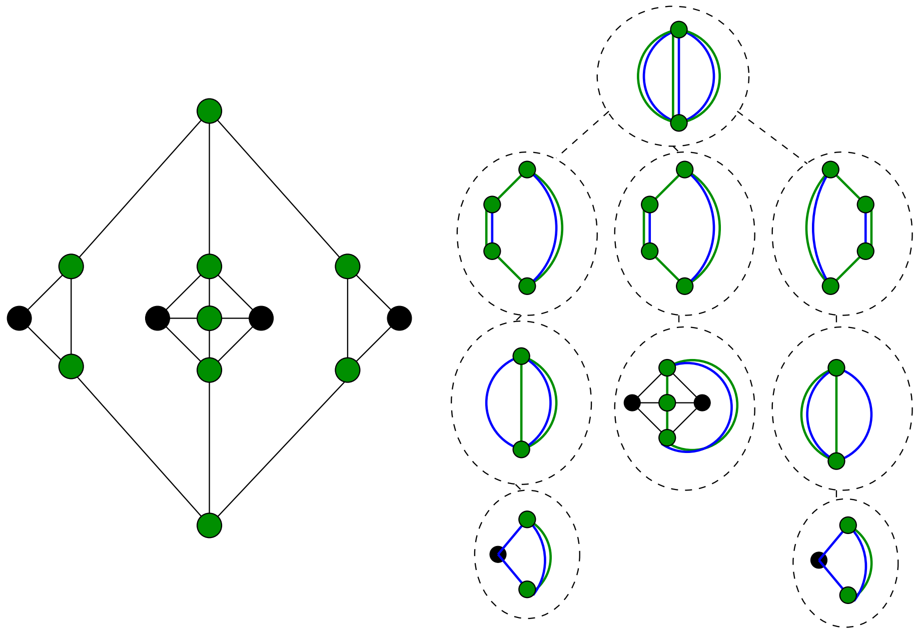


Figure 7.9: Continuation of Figure 7.6: After calling the algorithms `MarkEdgesI` and `MarkEdgesII`; ν-paths visualized green, ν-others visualized blue

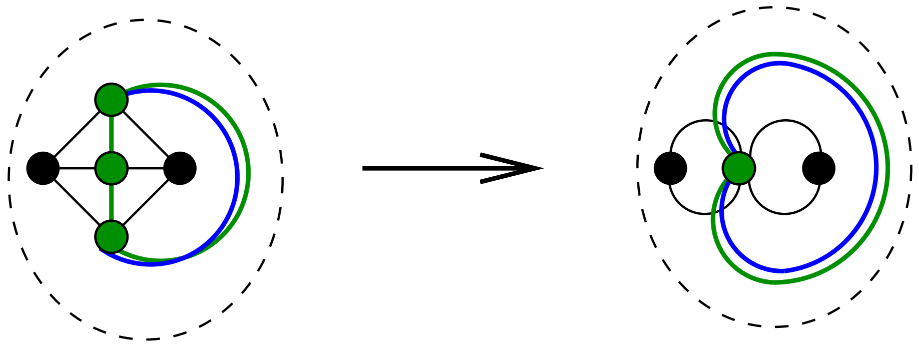


Figure 7.10: Continuation of Figure 7.6: Testing feasibility of the skeleton of the R-node; ν-paths visualized green, ν-others visualized blue

Algorithm 16: Part II: Algorithm `MarkEdgesII`.

```
Input: A $c$-connected planar clustered graph $C = (G, T)$, $G$ biconnected, results of
       MarkEdgesI
Result: all edges are marked lca($s$,$t$)-path or lca($s$,$t$)-other
$\mathcal{T}$ = SPQR-tree of $G$;
$s$,$t$ = pole vertices;
forall the nodes $v$ of $\mathcal{T}$ top downlevel by level do
    $S_f$ = Skeleton of father of $v$;
    $S$ = Skeleton of $v$;
    if father of $v$ is an S- or R-node then
        Calculate lca($s$,$t$)-path in $S_f$ other than twin edge;
        if lca($s$,$t$)-path exists then
            Add attribute lca($s$,$t$)-path to reference edge of $S$;
        if exists edge $e$ other than twin edge and not of lca($s$,$t$) in $S_f$ then
            Add attribute lca($s$,$t$)-other to reference edge of $S$;
    if father of $v$ is a P-node then
        if exists edge marked lca($s$,$t$)-path other than twin edge then
            Add attribute lca($s$,$t$)-path to reference edge of $S$;
        if exists edge marked lca($s$,$t$)-other other than twin edge then
            Add attribute lca($s$,$t$)-other to reference edge of $S$;
```

Algorithm 17: Part III: Algorithm `FeasibilityCheck` checks whether a given clustered graph is c-planar

```
forall the P-nodes do
    if at least three edges are marked both lca($s$,$t$)-path and lca($s$,$t$)-other then
        return false;
forall the R-nodes do
    Apply Algorithm Test_cplanar_triconnected;
return true;
```

because the size of the SPQR-tree of a planar biconnected graph including all skeletons is linear in the size of the graph [22].

We get the following algorithm for testing c-planarity of a given c-connected planar clustered graph $C = (G, T)$ with G biconnected.

Algorithm 18: Testing c-planarity of a c-connected clustered graph $C = (G, T)$ with G biconnected.

```
if $C$ is planar and $c$-connected then
    MarkEdgesI;
    MarkEdgesII;
    FeasibilityCheck;
```

An embedding algorithm can easily be constructed by using the same techniques as in the embedding step of SIPCA in Chapter 6:

In the positive case, we know that there are no isolating cut clusters. We have first to sort the virtual edges of the skeleton of each P-node. Those have to be ordered consecutively for each cluster - this can be done by solving the consecutive one's problem with the data structure *PQ*-tree [8]. Additionally, we might have to swap some skeletons of the R-nodes in order to not achieve an embedded isolating cut cluster (at least one edge in the skeleton that belongs to the lowest common ancestor of the poles has to be embedded in the outer face of the skeleton). This can be done in linear time.

7.1.3 The Algorithm for the Connected Case

Let $C = (G, T)$ be a c-connected planar clustered graph with G connected but not biconnected. Therefore, G has cut vertices that connect blocks.

Obviously, we have to check for two connected components B_1 and B_2 with a common cut vertex c whether there is a planar embedding with a face f such that f has

1. c on its boundary and
2. one boundary of f is parted into a path P_1 from c to c using edges of B_1 and a path P_2 from c to c using edges of B_2 and
3. the lowest of all lowest common ancestors of the end vertices of the edges of P_1 or P_2, respectively, is lower or equal to the cluster that contains c.

We build a BC-tree, where B and C are the vertices of the tree, representing B the blocks and C the cut vertices.

For each block we check whether there are at least two clusters and mark the corresponding B-node in the tree. For each cut vertex that belongs to such a block we mark it red and assign to it the lowest cluster of T that is included in the block.

We construct for each block B a SPQR-tree with the red marked cut vertices. Whenever a red marked vertex occur in the MarkEdges procedure, we add the attribute lca(s,t)-other with s, t the pole vertices of the corresponding skeleton.

Finally, in the R-nodes for each red marked vertex c we check whether there is a face bounded by c that has an edge on its boundary with the lowest common ancestor of its end vertices equal or lower to the cluster of c.

7.2 co-Connected Clustered Graphs

Let $C = (G, T)$ be a co-connected planar clustered graph. In this chapter we show that in this case c-planarity is decidable and realizable in polynomial time.

Theorem 7.6. *Let a co-connected planar clustered graph $C = (G, T)$ be given. C is c-planar if and only if it has a c-connected planar super clustered graph.*

Proof. If C is c-planar it has a c-planar c-connected super-clustered graph SC. Therefore, SC is planar. Let a c-connected planar super-clustered graph be given that contains a co-connected planar clustered graph. Then SC is co-connected. Since SC is planar, it is c-planar. Then C is also c-planar. □

Hence our aim is to find an edge-augmentation that makes C c-connected if one exists such that C remains planar. Therefore, we can state the following corollary.

Corollary 7.1. *Let a co-connected planar clustered graph $C = (G, T)$ be given. C is c-planar if and only if a c-connected super-clustered graph of C exists that has no minor isomorphic to $K_{3,3}$ or K_5.*

Proof. Since C is co-connected the super-clustered graph has to be completely connected. Additionally, the super-clustered graph is not allowed to have a minor isomorphic to $K_{3,3}$ or K_5, therefore is planar. Recall that a completely connected planar clustered graph is c-planar [15, 54]. By Theorem 2.4, then C is c-planar.

If C is c-planar, it has a c-planar embedding that can be extended to a completely connected planar clustered graph. □

Furthermore, since C is co-connected, it has no cut cluster.

In this chapter, first we discuss c-planarity of a subclass of planar co-connected clustered graphs with the additional property: For any pair of disjoint clusters ν, μ of C $G - (G(\nu) \cup G(\mu))$ is connected.

In the following, we show how to compute a c-planar embedding of this subclass, if one exists, in polynomial time. Furthermore, using this results, we discuss c-planarity of general co-connected planar clustered graphs.

7.2.1 The Algorithm for the Triconnected Case

Since G is triconnected G has a unique planar embedding Π beside mirroring.

We choose the planar embedding Π that has an edge of the root cluster in the boundary of the outer face.

Since for all clusters ν $G - G(\nu)$ is connected C has no cut cluster. Furthermore, we cannot create a cut cluster by edge-augmentation.

Assume that we have two vertices v_1 and v_2 of a cluster ν. Furthermore, we assume that v_1 and v_2 belong to different connected components of $G(\nu)$ (see Figure 7.47).

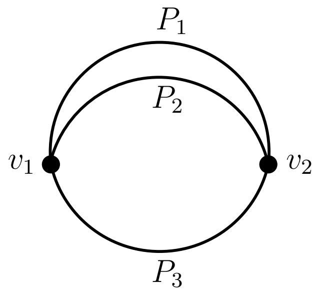


Figure 7.11: Triconnected case: connectivity considerations

Since G is triconnected there exist three disjoint paths from v_1 to v_2, say P_1, P_2 and P_3. Further, since v_1 and v_2 are not connected in $G(\nu)$ P_1, P_2 and P_3 use vertices of $G - G(\nu)$. Since $G - G(\nu)$ is connected there has to be a path connecting P_1, P_2 and P_3 in $G - G(\nu)$, say P_4 (see Figure 7.48).

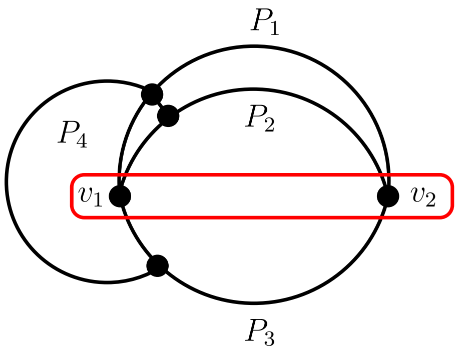


Figure 7.12: Triconnected case: connectivity considerations

If we have two vertices w_1 and w_2 of a sibling cluster μ there are not so many possibilities how they can be placed in our construction. Figures 7.13 to 7.16 give some examples in which $G-(G(\nu) \cup G(\mu))$ is connected (green path assures connectivity of $G-G(\mu)$ such that $G-(G(\nu) \cup G(\mu))$ is connected). In Figures 7.17 to 7.20 the connection of $G-(G(\nu) \cup G(\mu))$ is skipped.

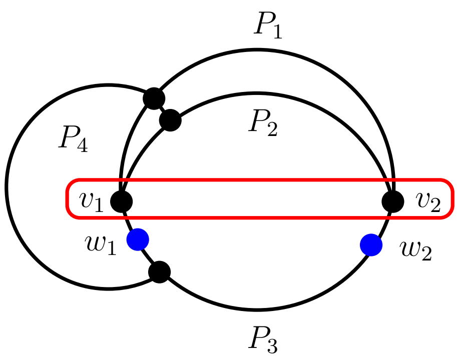


Figure 7.13: Triconnected case: connectivity considerations

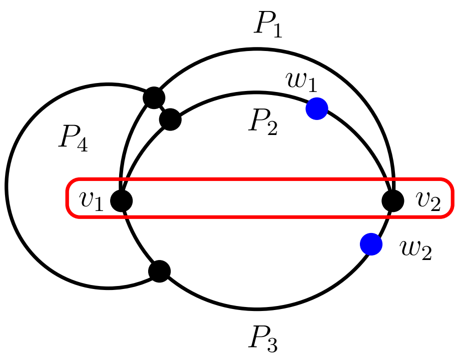


Figure 7.14: Triconnected case: connectivity considerations

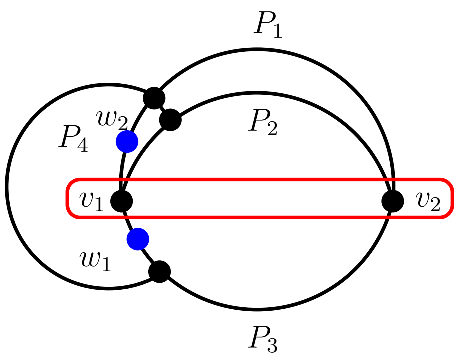


Figure 7.15: Triconnected case: connectivity considerations

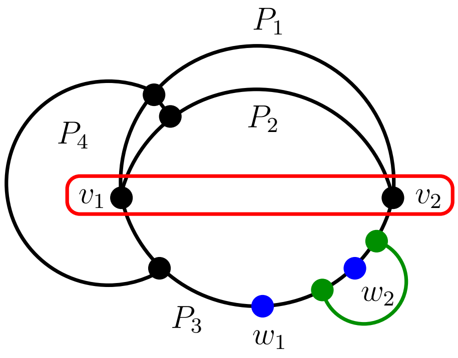


Figure 7.16: Triconnected case: connectivity considerations

Furthermore, if we omit that $G - (G(\nu) \cup G(\mu))$ is connected we may place w.l.o.g. w_1 on path P_4 and w_2 on all other paths P_2, P_3 such that $G - G(\mu)$ is connected (see Figures 7.17 to 7.20). Again, since G is triconnected, the edge-augmentation is feasible if there is no conflict in any face for any two sibling clusters ν, μ.

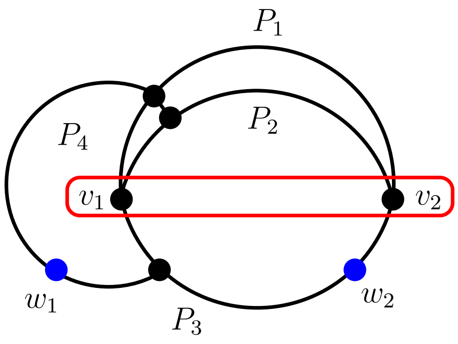


Figure 7.17: Triconnected case: connectivity considerations

Consequently, if we have any conflicts between two clusters in a face of Π, an example is visualized in Figure 7.21, C is not c-planar. In Figure 7.22 an example is given where the co-connected clustered graph is c-planar.

We can state the following theorem.

Theorem 7.7. *Let a co-connected planar clustered graph $C = (G, T)$ be given. Let G be triconnected.*

C is c-planar if and only if

1. *its planar embedding Π is conflict-free in each face for every cluster and*
2. *there exists a SIPCA for each cluster.*

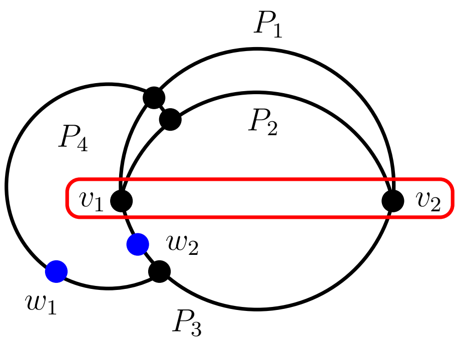


Figure 7.18: Triconnected case: connectivity considerations

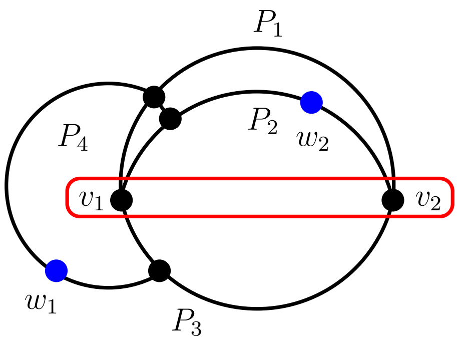


Figure 7.19: Triconnected case: connectivity considerations

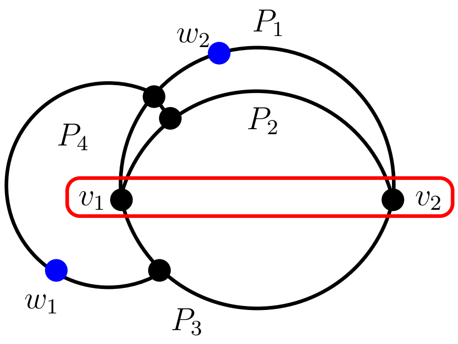


Figure 7.20: Triconnected case: connectivity considerations

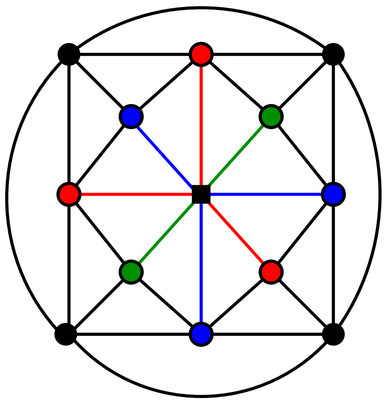


Figure 7.21: A co-connected clustered graph with cluster conflicts in a face.

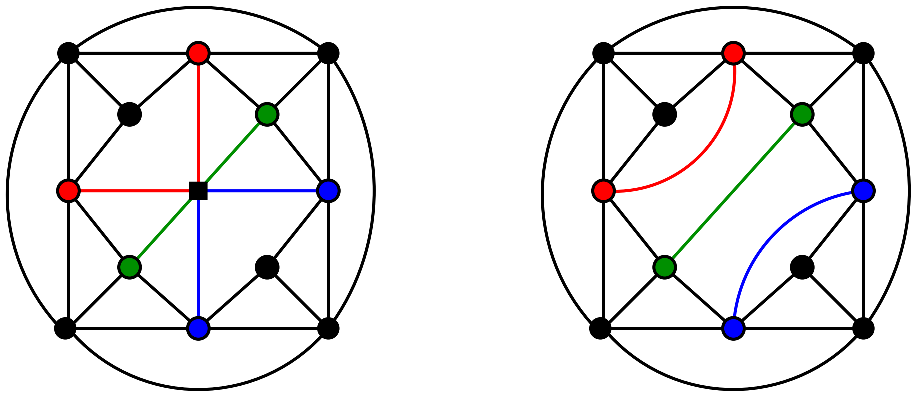


Figure 7.22: A co-connected clustered graph with no cluster conflicts.

Proof.

1. Assume that the planar embedding has conflicts between vertices of sibling clusters in the faces. Then ν has to be a cluster cut vertex otherwise C is not c-planar. Then $G - G(\nu)$ is not connected that contradicts the assumption that C is co-connected.

2. If C has a planar embedding without the mentioned conflicts then C has a c-connected planar super-clustered graph that is c-planar. Then C is also c-planar.

□

By Theorem 7.7, we have to check if each face of Π is conflict-free.

Consequently, there exists an edge-augmentation for every face of Π such that C is c-connected if and only if C has a c-connected planar super-clustered graph. In particular, C is c-planar if and only if we apply a SIPCA (see Chapter 6) to each cluster of C in Π and the resulting planar clustered graph is additionally c-connected. Observe that since no cluster conflicts are allowed we can apply a SIPCA independently to each cluster. Since we are interested in a polynomial running time algorithm, we modify the SIPCA approach such that we augment Π by visiting each face at most a linear time. This can be done very easily, see Algorithm 19 for a pseudo code.

7.2.2 The Algorithm for the Biconnected Case

Assume G is biconnected but not triconnected. Further, we assume that for any pair of sibling clusters ν, μ of C $G - (G(\nu) \cup G(\mu))$ is connected.

Firstly, for simplification we suppose that G is series-parallel. Figure 7.23 gives an example: in this case for each cluster ν of C there is a vertex in each path between two split pairs in G. Since for each cluster ν $G - G(\nu)$ is connected, we observe that vertices of each cluster ν are ordered naturally near to each other. In particular, the vertices of a cluster ν are leveled in G and hence every cluster appears in a planar hierarchical layering in C.

Naturally, the question arises if we might expect this for every planar embedding of G. Unfortunately, the answer is negative. An example is given in Figure 7.38. In this case, the vertex v_1 cannot be connected to a red colored vertex in the given embedding since this would result in an edge-region crossing with a blue colored cluster. Fortunately, a feasible planar embedding can be constructed by embedding v_1 in the other face that has a red colored vertex without such an edge-region crossing. Therefore, even if we might connect a red colored vertex with an other red colored vertex (v_1) in at least two different faces only one of those is feasible. Somehow the connection between the blue colored vertices seems to have a priority: after connecting the blue colored vertices, there is only one face to embed v_1 with an other red colored vertex.

We discuss the observations made so far in the following.

Algorithm 19: Algorithm `Test_cplanar_triconnected` that tests c-planarity for a co-connected clustered graph $C = (G, T)$ with G planar and triconnected.

```
Input: A co-connected clustered graph $C = (G, T)$, $G$ planar and triconnected
Result: true, if and only if $C$ is $c$-planar, otherwise false.
Choose a planar embedding $\Gamma$ with an edge $e$ on the boundary of the outer face that
belongs to the root cluster;
forall the clusters $\nu$ do
    first($\nu$)←nil;
forall the vertices $v$ do
    Assign $v$ to the cluster $\nu$ that contains $v$ and is farest from the root cluster in
    $T$;
forall the faces $f$ do
    Mark vertices $v$ on the boundary of $f$: prev(cluster of $v$)←nil;
forall the faces $f$ do
    prev←nil;
    forall the vertices $v$ on its boundary do
        if first(cluster of $v$)=nil then
            first(cluster of $v$)← $v$;
        if prev(cluster of $v$)=nil then
            prev(cluster of $v$)← $v$;
        else
            Add edge $e$ =(prev(cluster of $v$),$v$) to $f$;
            Assign edge $e$ to lca(prev(cluster of $v$),$v$);
            prev(cluster of $v$)← $v$;
            if lca(prev,$v$)$\neq$root cluster then
                Add edge $e$ =(prev,$v$) to $f$;
                Assign edge $e$ to lca(prev,$v$);
        prev← $v$;
Treat modified clustered graph as $C$;
if $C$ is not $c$-connected or not planar then
    return false;
return true;
```

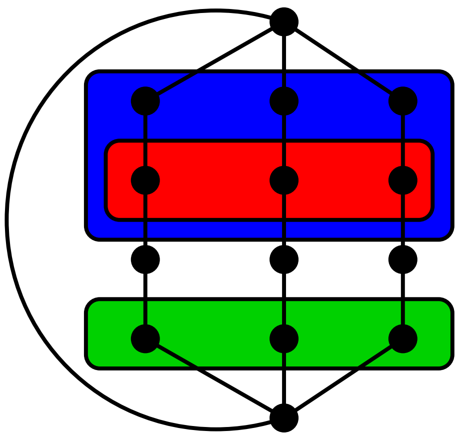


Figure 7.23: A co-connected planar clustered graph with a series-parallel underlying graph

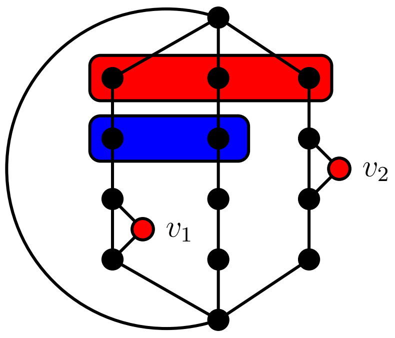


Figure 7.24: A planar embedding of a co-connected planar clustered graph that is not c-planar.

Assume that we have two vertices v_1 and v_2 of a cluster ν. Furthermore, we assume that v_1 and v_2 belong to different connected components of $G(\nu)$ (see Figure 7.25).

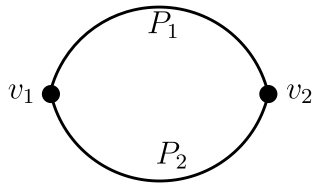


Figure 7.25: Biconnected case: connectivity considerations

Since G is biconnected there exist two disjoint paths from v_1 to v_2, say P_1 and P_2. Further, since v_1 and v_2 are not connected in $G(\nu)$ P_1 and P_2 use vertices of $G - G(\nu)$. Since $G - G(\nu)$ is connected there has to be a path connecting P_1 and P_2 in $G - G(\nu)$, say P_3 (see Figure 7.26).

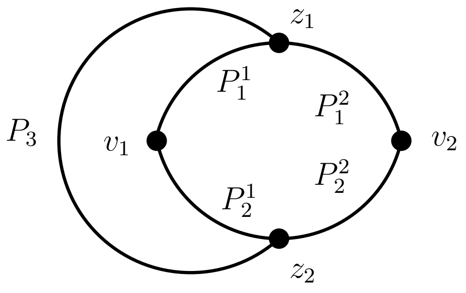


Figure 7.26: Biconnected case: connectivity considerations

Consequently, v_1 and v_2 are no split pairs of G. More precisely, there is no pair of vertices that belong to different connected components of a disconnected cluster (clearly other than the root cluster) that is a split pair of G. Let $z_1 \in P_1 \cap P_3$ and $z_2 \in P_2 \cap P_3$. If we connect v_1 and v_2 by an edge e we introduce a minor isomorphic to K_4. Intuitively this is clear since we introduce a third path from v_1 to v_2 that is disjoint to P_1 and P_2. Hence, after augmentation v_1, v_2, z_1 and z_2 is contained in a triconnected minor. Consequently, P_3 and e cannot be embedded in exactly one face of cycle $Z = P_1 \cup P_2$. We say that P_3 and e are *conflicting*. We split P_i $(i = 1, 2)$ in two paths each: P_i^1 from v_1 to z_i and P_i^2 from z_i to v_2. Furthermore, z_1 and z_2 are contained in $G - G(\nu)$ by construction. Therefore, we have the situation visualized in Figure 7.27.

We may assume that all vertices in P_1^i and P_2^i that belong to ν are contained in the same connected component of $G(\nu)$ as v_i for $i = 1, 2$ since we may apply the previous arguments recursively.

Observe that if we introduce an other path that is conflicting with P_3 *and* e we introduce a minor isomorphic to $K_{3,3}$. Consequently, if we have two vertices w_1 and w_2 of a sibling

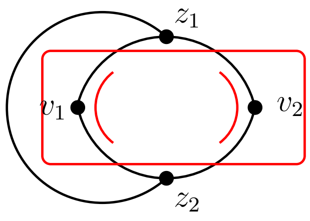


Figure 7.27: Biconnected case: connectivity considerations

cluster μ such that P_3 is contained in $G-G(\mu)$ and edge (w_1, w_2) is conflicting with e and P_3 our construction is not c-planar. Consequently, the construction is not c-planar if $w_1 \in P_1^1$ or $w_1 \in P_1^2$, respectively, and $w_2 \in P_2^2$ or $w_1 \in P_2^1$, respectively (see Figure 7.28). Observe that by deleting P_3 and z_1, z_2 being a split pair both of ν and μ become a cut cluster. Therefore, in this case sibling clusters are not allowed to be conflicting in the mentioned way.

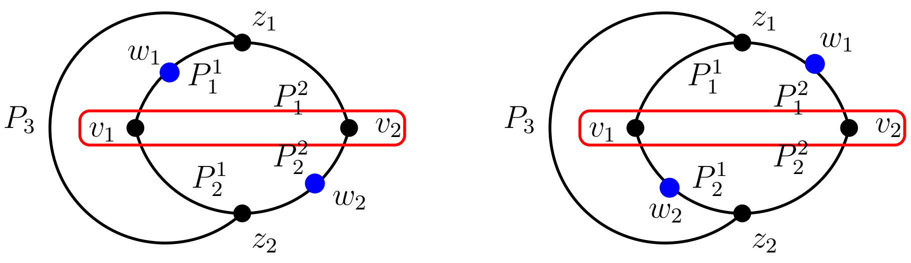


Figure 7.28: Biconnected case: connectivity considerations

Hence, the construction is c-planar if $w_1 \in P_1^1$ or $w_1 \in P_2^1$, respectively, and $w_2 \in P_1^2$ or $w_1 \in P_2^2$, respectively (see Figure 7.29).

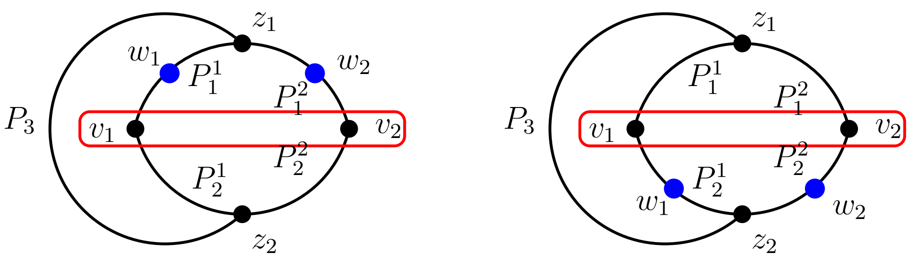


Figure 7.29: Biconnected case: connectivity considerations

In this case e' is conflicting with P_3 but not with e.

As already mentioned before, P_3 is contained in $G-G(\nu)$ and in $G-G(\mu)$. Therefore, by deletion of P_3 and z_1, z_2 being a split pair ν and μ become each a cut clusters in this case. Consequently, edge (v_1, v_2) is not allowed to be conflicting to edge (w_1, w_2). Observe that in this case *every* planar embedding has no conflicting clusters. Hence, we get a hierarchical structure as visualized in Figure 7.30 that was already observed for a c-planar drawing by Feng [29].

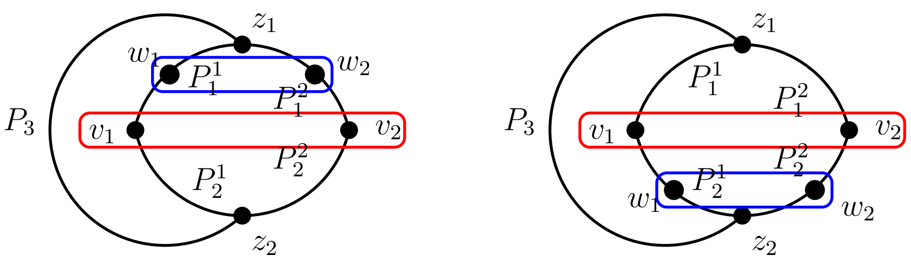


Figure 7.30: Biconnected case: connectivity considerations

Further, we observe that we might assume in the following that z_1 and z_2 belong to $G-G(\mu)$ since otherwise we can assume that there is an other path that connects the two disjoint paths between w_1 and w_2. An example is visualized in Figure 7.31, the path P_4 is colored green and might share a vertex with P_3. Then, in the negative case our construction is c-planar (what does not mean that it is c-planar in general in the following since there might be additional paths in a planar graph which are conflicting).

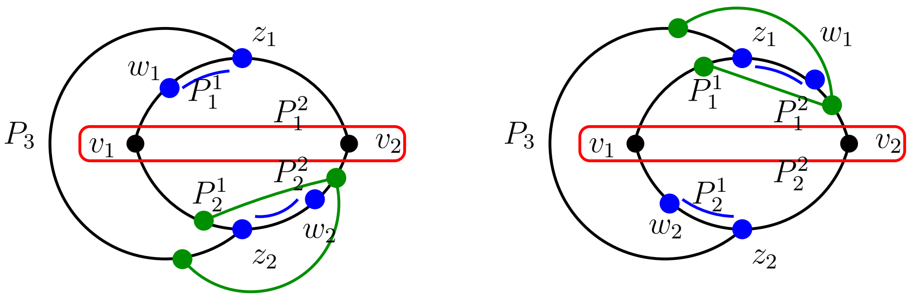


Figure 7.31: Biconnected case: connectivity considerations

Additionally, observe that once one of z_1 or z_2 is contained in μ and the paths P_3, P_4 are conflicting as visualized in Figure 7.31, our construction is c-planar. Once P_3 and P_4 share a vertex as visualized in Figure 7.32 our construction is not c-planar.

On the other hand, the construction is c-planar if $w_1 \in P_1^1$ or $w_1 \in P_1^2$, respectively, and $w_2 \in P_2^1$ or $w_1 \in P_2^2$, respectively (see Figure 7.33).

Unfortunately, the construction is not co-connected. Consequently, there exists a path P_4 that connects the two disjoint paths between w_1 and w_2 in $G-G(\mu)$ (see Figure 7.34, path is visualized green). Observe that P_4 is not conflicting with e or P_3.

Additionally, in this case $e' = (w_1, w_2)$ is conflicting with e but not with P_3. Therefore, the construction is c-planar, but not every planar embedding of this construction is c-planar as visualized in Figure 7.35.

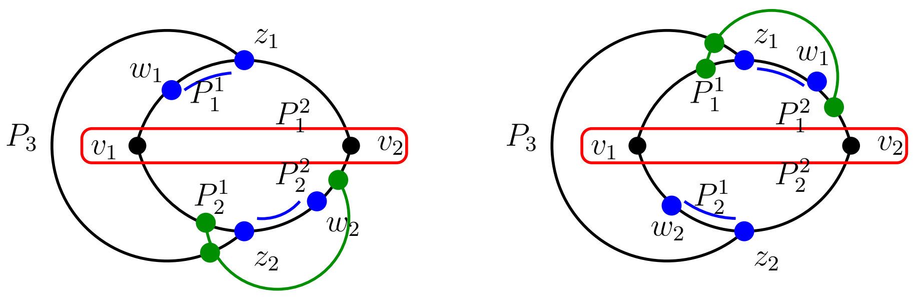


Figure 7.32: Biconnected case: connectivity considerations

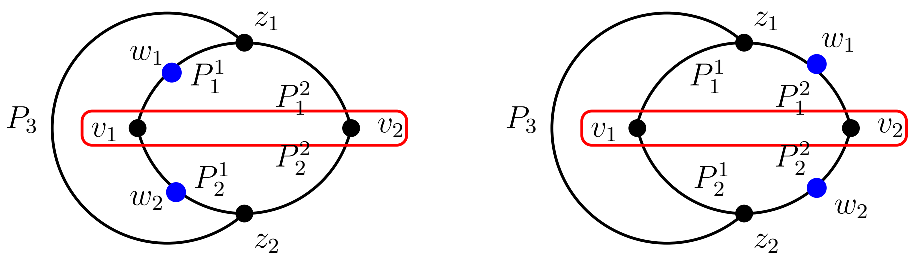


Figure 7.33: Biconnected case: connectivity considerations

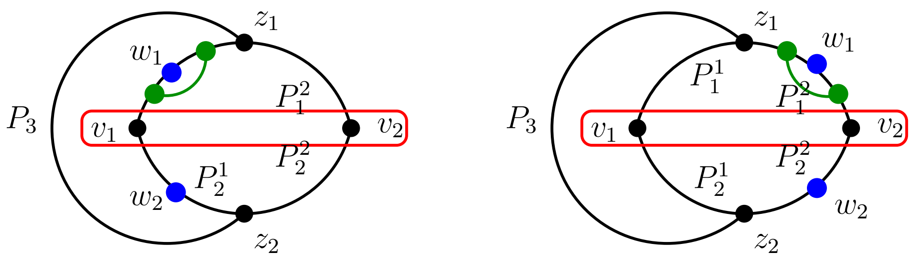


Figure 7.34: Biconnected case: connectivity considerations

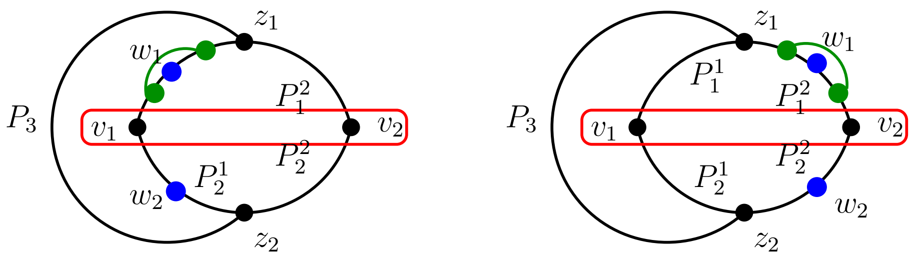


Figure 7.35: Biconnected case: connectivity considerations

If we combine our observations an additional critical situation arises (see Figures 7.36 and 7.37). In Figure 7.37 the bold edges are the original edges and the dashed are the added edges.

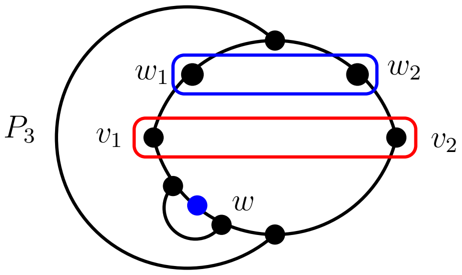


Figure 7.36: Biconnected case: connectivity considerations

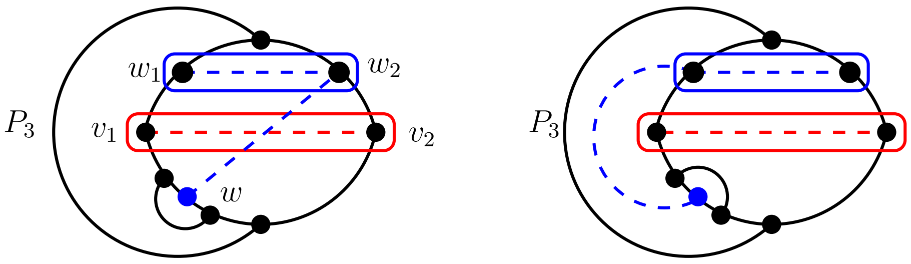


Figure 7.37: Biconnected case: connectivity considerations

In the case of Figure 7.30 we already observed that the edge-augmentation with (v_1, v_2) and (w_1, w_2) is feasible and results in a c-planar embedding. In the new case the blue colored vertex v also belongs to the blue colored cluster and has to be connected either to w_1 or w_2. Figure 7.37 shows the two possibilities for edge-augmentation: the first between w and w_2 and the second between w and w_1. On the left-hand side of the Figure 7.37 the edge-augmentation cause an edge-region crossing with the red colored cluster. Observe that the red colored and dashed edge (v_1, v_2) already is conflicting with path P_3. Therefore, this situation creates a minor isomorphic to $K_{3,3}$. The right-hand side of Figure 7.37 shows an edge-augmentation between w and w_1. Observe that edge (w, w_1) is not conflicting with P_3 but conflicting with (v_1, v_2). Hence, the construction is still c-planar after edge-augmentation. Consequently, only the edge-augmentation between w and w_1 is feasible and results in a c-planar embedding.

Finally, we observe that a child cluster ν shares the same path as its parent cluster $pa(\nu)$ in $G - G(pa(\nu))$.

In the case that we omit the additional condition that for any pair of sibling clusters ν, μ $G - (G(\nu) \cup G(\mu))$ is connected we observe that there arise an other critical situation. An example is visualized in Figure 7.38.

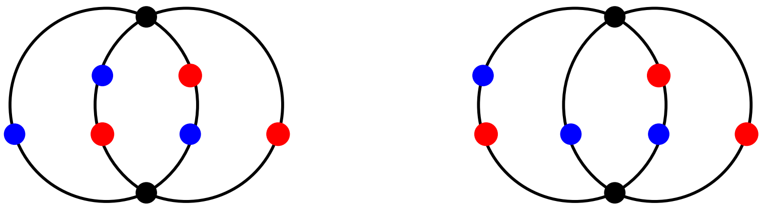


Figure 7.38: A co-connected planar clustered graph C with underlying series-parallel graph and two sibling clusters, highlighted with blue and red color: left-hand side a planar embedding of C that is not c-planar, on the right-hand side a planar embedding of C that is c-planar

In comparison to the case visualized in Figure 7.30, in this case if C is c-planar not all planar embeddings are c-planar. In this case if C is c-planar there exists *a* planar embedding where the sibling clusters are not conflicting (otherwise C would have a cut cluster and consequently be not co-connected).

However, we may observe the following fact in this case for a c-planar clustered graph $C = (G, T)$ (see Figure 7.30). Once any pair of clusters, say ν and μ, disconnects the underlying graph G by their removal *and* there exists a planar embedding of C such that there is a face where ν and μ are conflicting, then there exists an additional planar embedding with the following properties ($S = \{\nu, \mu\}$):

- $G - G(\nu)$ and $G - G(\mu)$ function as a changeover to avoid crossing conflicts between ν, μ.
- Vertices of $G(\gamma_1) - (G - G(\gamma_2))$ with $\gamma_1 \in S$ and $\gamma_2 \in S - \{\gamma_1\}$ may be sorted into two vertex sets such that vertices of each such vertex set appear hierarchically in respect to γ_1 and γ_2, respectively, in the final c-planar embedding.

Consequently, once we are able to *sort* the vertices of any cluster pair ν, μ in the previous described way and the constructed planar embedding has no face that is conflicting for ν, μ then our given clustered graph is c-planar. We will have a closer look how to do this sorting to achieve such an embedding (if one exists) in Section 7.2.2.

By induction, we formalize our observations.

Observation 7.1. *For any additional edge between a pair v, w of vertices of a cluster ν (notice: v, w are not connected in $G(\nu)$) there is a conflicting path in $G - G(\nu)$ in respect to cycle Z of the two disjoint paths connecting v and w. Consequently, any additional edge (v, w) is contained in a triconnected subdivision (even stronger: minor) of the resulting G.*

Observation 7.2. *Clearly, in order to keep the planarity of G no additional conflicting path is allowed to be included otherwise a $K_{3,3}$ minor is constructed.*

Observe that v, w form a split pair by deleting its conflicting path in $G - G(\nu)$. More precisely, the cluster that contains v, w turns into a cut cluster.

Observation 7.3. *Let a planar co-connected clustered graph $C = (G, T)$ be given. Let G be biconnected but not triconnected and s, t be a split pair of G. Let μ be the lowest common ancestor of s, t in T. Let K be the set of the connected components after deletion of s,t. Then either*

1. *μ is equal to the root cluster or*
2. *$G - G(\mu)$ belongs to exactly one connected component k of K (that might also contain additional vertices of $G(\nu)$).*

Summarizing all observations we might expect that once we find a planar embedding with no conflicting sibling clusters we have a c-planar embedding at hand.

Fortunately, the following theorem gives us the guarantee that at least one such planar embedding of G exists if and only if C is c-planar.

Theorem 7.8. *Let a co-connected planar clustered graph $C = (G, T)$ be given. Let G be biconnected.*

C is c-planar if and only if

1. *there exists a planar embedding Π of C such that each face has no conflict between vertices of non-descendant clusters and*
2. *there is a SIPCA for each cluster in Π.*

Proof.

1. Assume that there is no such planar embedding. Then every planar embedding has a conflict between vertices of sibling clusters. Furthermore, assume that C is c-planar. Then there exists an assignment of one cluster for each conflict such that for each cluster ν $G(\nu)$ can be connected and C remains c-planar. Then ν is a cut cluster in the original clustered graph C (otherwise C would not be c-planar). Then $G - G(\nu)$ is not connected that contradicts the assumption that C is co-connected. Therefore, there is at least one planar embedding that satisfies the condition of the theorem.
2. If a planar embedding Π of C exists in which each face has no conflict between vertices of sibling clusters, C is co-connected, and for every cluster exist a SIPCA then C has a planar c-connected super-clustered graph. Then C is obviously c-planar.

□

By Theorem 7.8 we are seeking for an algorithm that finds such a planar embedding if and only if one exists. Whenever we consider the case where C is co-connected such that for any pair of sibling clusters ν, μ of C $G-(G(\nu)\cup G(\mu))$ is connected, our consideration of feasible edge-augmention results that we may augment each cluster independently but there is one edge-augmention that is forbidden to use, visualized on the left-hand side of Figure 7.37. Hence, whenever this situation occurs the edge-augmentation on the right-hand side of Figure 7.37 has to be applied. An example is given in Figure 7.39.

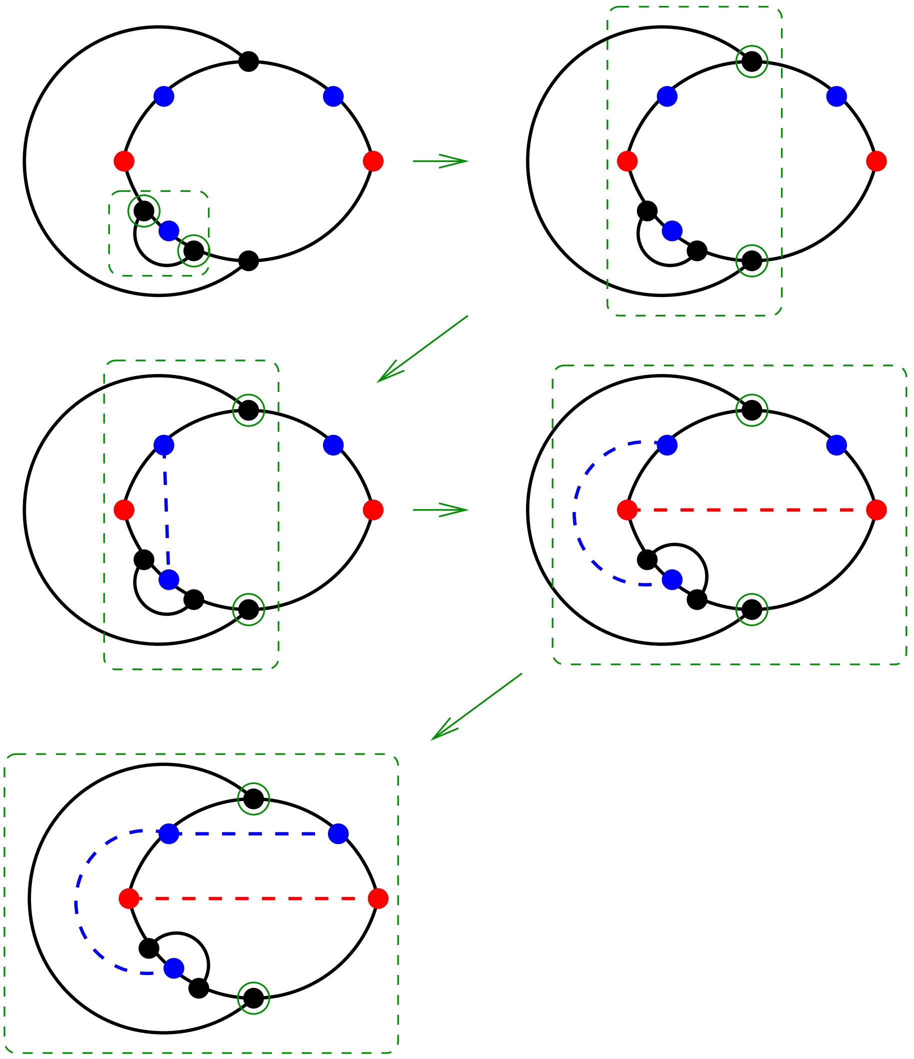


Figure 7.39: Biconnected case: an example, green circled vertices belong to a split pair, added edges are visualized with a dashed line, the subgraph involved in the current edge-augmentation is highlighted with a green dashed rectangle

In the more general case when we omit the previous additional condition we observed that we have to take more special cases into account, an example is visualized in Figure 7.37.

In the following we consider an algorithm that tests c-planarity for this more general case.

We adapt the SIPCA approach (see Chapter 6) for all clusters of C. For this, we use the SPQR-tree data structure (see Chapter 2) to get a polynomial running time algorithm. Again, we split the approach into two steps, i. e. parted into the coloring (see Section 7.2.2) and the embedding algorithm (see Section 7.2.2).

The Coloring Algorithm

We adapt the terminology defined in Chapter 6 for our purposes. For a cluster ν, we call a vertex *ν-blue*, if the vertex is contained in ν and *ν-black* otherwise. For each cluster ν, we assign one of two colors to each edge in each skeleton: ν-blue or ν-black. We call an edge in a skeleton ν-blue, if its expansion graph contains vertices of ν and ν-black otherwise.

Additionally, for each cluster ν we assign the attribute *ν-permeable* to some ν-blue edges. Intuitively, an edge is ν-permeable if we can construct a path connecting only ν-blue vertices through its expansion graph without destroying any other possibility for connection of any sibling cluster μ. Let $G(e)$ be the expansion graph of edge e in skeleton $\mathcal{S}$. In any planar embedding $G(e)$, there are exactly two faces that have e on their boundary. This follows from the fact that in a planar biconnected graph, every edge is on the boundary of exactly two faces in every embedding. We call the edge e in $\mathcal{S}$ ν-permeable, if there is an embedding Π of $G(e)$ and a list of at least two faces $L = (f_1, \ldots, f_k)$ in Π that satisfies the following properties:

1. The two faces f_1 and f_k are the two faces with e on their boundary.
2. For any two faces f_i, f_{i+1} with $1 \le i < k$, there is a ν-blue vertex on the boundary between f_i and f_{i+1}.
3. there is no conflicting path of a sibling cluster.

We call a skeleton $\mathcal{S}$ of a node v of $\mathcal{T}$ ν-permeable if the pertinent graph of v together with the virtual edge of $\mathcal{S}$ have the two properties stated above. Therefore $\mathcal{S}$ is ν-permeable if the twin edge of its virtual edge is ν-permeable. We call an expansion graph $G(e)$ ν-permeable if it has an embedding Π and a list of at least two faces $L = (f_1, \ldots, f_k)$ that satisfies the two properties stated above.

We develop an algorithm that marks for each cluster ν each edge in every skeleton of the SPQR-tree $\mathcal{T}$ of G with the colors ν-black or ν-blue and assign the attribute ν-permeable depending on the expansion graph of the edge. The algorithms work recursively. We assume that $\mathcal{T}$ is rooted at node r and r is not a Q-node.

For each cluster ν, we mark all the edges of the skeletons of the children of r recursively either ν-black or ν-blue and assign the ν-permeable attribute by treating them as the roots of subtrees. Each edge in the skeleton $\mathcal{S}$ of r except the reference edge corresponds to a child of r. Let e be such an edge in $\mathcal{S}$, v the corresponding child of r and $\mathcal{S}'$ the skeleton of v. If $\mathcal{S}'$ contains a ν-blue edge or vertex, we mark e ν-blue and otherwise ν-black.

Let $\mathcal{T}'$ be the resulting SPQR-tree and v a node of $\mathcal{T}'$.

For each cluster ν, the ν-permeability of e depends on the type of v:

Q-node: We mark e ν-permeable if the skeleton $\mathcal{S}'$ contains a ν-blue vertex.

S-node: We take a copy $\mathcal{S}''$ of the skeleton $\mathcal{S}'$. Then we connect the source of each ν-blue edge (that we mark connected) to the source of a ν-permeable edge or ν-blue vertex if one exist. If the resulting skeleton $\mathcal{S}''$ is not planar C is not c-planar and we stop. Otherwise, if the resulting skeleton $\mathcal{S}''$ contains a ν-blue vertex or a ν-permeable edge, we mark e ν-permeable.

P-node: If the skeleton $\mathcal{S}'$ contains only ν-permeable edges or a ν-blue vertex, we mark e ν-permeable.

R-node: We take a copy $\mathcal{S}''$ of the skeleton $\mathcal{S}'$.

Firstly, we apply the algorithm for the triconnected case (see Algorithm 19) on $\mathcal{S}''$ for all ν-blue vertices and for all sources of the ν-permeable edges for every cluster ν. If the algorithm fails, C is not c-planar. In the positive case, we split the existing faces such that all ν-blue vertices and all ν-permeable edges get connected.

Secondly, we connect all ν-blue edges to the augmentation previously made for every cluster ν. We do this by connecting the source of each ν-blue edge e (that we mark connected) to the source of a ν-permeable edge or ν-blue vertex if one exist in one of the neighboring faces of e.

Since this augmentation cannot be done independently for each cluster ν, we transform the problem to a 2-SAT instance, that is known to be solvable very efficiently in polynomial time, in the following way. As a preparation, we construct a conflict graph for each face f of the given planar embedding of $\mathcal{S}''$. The conflict graph represents the worst case scenario of conflicting cluster pairs that may occur in f. More particular, if we would use the Algorithm 19 for the triconnected case in f such that all ν-blue edges for all clusters ν that are contained on the boundary of f has to be reached within f, certain connection edges might cross within f. For each such edge we add a vertex in the conflict graph and whenever two connection edges cross in f we connect the corresponding vertices of the connection edges in the conflict graph. Clearly, if the resulting conflict graph is not bipartite, we are done because C cannot be c-planar since it would not be planar after augmentation.

Otherwise, we assign to each connection edge e in f a boolean variable x_e^f, that we set 1 if e should be embedded within f and 0 otherwise. We transform each conflict graph into a 2-SAT subinstance. Intuitively spoken, if two connection edges e_1, e_2 cross in a face f then obviously *either* e_1 *or* e_2 may be embedded within f for planarity reasons. Hence we assign for each edge $e = (x_{e_1}^f, x_{e_2}^f)$ of the conflict graph of f

$$x_{e_1}^f \vee x_{e_2}^f$$

$$\bar{x}^f_{e_1} \vee \bar{x}^f_{e_2}$$

to our 2-SAT instance.

Additionally, every ν-blue edge e for a cluster ν has two neighboring faces, say f and g. In the best case, vertices of ν that belong to the pertinent graph of e can be embedded *either* within f *or* within g. Hence we assign

$$x^f_e \vee x^g_e$$
$$\bar{x}^f_e \vee \bar{x}^g_e$$

to our 2-SAT instance for each ν-blue edge of a cluster ν.

It is known that the (constructed) 2-SAT instance can be solved very efficiently using the strong connectivity of directed graphs. If the 2-SAT instance is unsatisfiable then C is not c-planar and we are done.

We assume next that we get a satisfiable solution of our 2-SAT instance. This solution indentifies that each ν-blue edge may be connected with a ν-permeable edge or ν-blue vertex on one of its neighboring faces *and* that none of those connections is crossing an other one. Consequently, $\mathcal{S}''$ remains planar after this edge-augmentation.

Therefore, we modify $\mathcal{S}''$ in the following way. We connect the source of each ν-blue edge e (that we mark connected) to the source of a ν-permeable edge or ν-blue vertex in the face f where x^f_e is set to 1 in the final solution.

Finally, we consider a graph H where the vertices are the faces of $\mathcal{S}''$ and there is an edge between two vertices if there is a ν-permeable edge or a ν-blue vertex on the boundary that separates the two faces. Let s and t be the two faces left and right of the virtual edge of $\mathcal{S}'$. If there is a path in H connecting s and t, we mark e ν-permeable.

After executing this algorithm, which we call `MarkEdgesPhase1` (see Algorithm 21 for a pseudo-code), all edges of the skeleton of the root node r are marked, because we have seen all the ν-blue vertices of the graph for each cluster ν. All other skeletons except the skeleton of r contain one edge that is not yet marked: the virtual edge of the skeleton.

The algorithm `MarkEdgesPhase2` (see Algorithm 22 for a pseudo-code) works top down by traversing $\mathcal{T}'$ from the root to the leaves. The edges of the skeleton of the root r of $\mathcal{T}'$ are already marked in the first step, therefore we can proceed to the children and mark the virtual edges of its children. Let v be a node in $\mathcal{T}'$ where the skeleton $\mathcal{S}'$ of the parent node is already completely marked. Let ν be a cluster of ν. We mark the virtual edge e in the skeleton $\mathcal{S}$ of v ν-blue if there is a ν-blue edge or vertex in the skeleton of the parent. The ν-permeability of e again depends on the type of the skeleton in exactly the same way as in the algorithm `MarkEdgesPhase1`. Note that the case Q-node is irrelevant here because the Q-nodes form the leaves of the tree.

Algorithm 20: Algorithm `findPath` checks if the expansion graph of the twin edge of an edge in an R-node skeleton is permeable.

```
Input: An edge $e$ and an $R$-node $v$ in $\mathcal{T}$ with the property that $e$ is contained in
       the skeleton $\mathcal{S}$ of $v$ and all edges in $\mathcal{S}$ are marked
Result: For every cluster $\nu$: returns $\nu$-permeable if the expansion graph of the twin
        edge of $e$ is $\nu$-permeable
Compute an arbitrary embedding $\Pi$ of $\mathcal{S}$;
Compute the dual graph $D$ of $\mathcal{S}$ with respect to $\Pi$;
foreach edge $e'$ of $D$ do
    if the primal edge of $e'$ is $\nu$-permeable then
        Shrink $e'$ in $D$;
Let $v_1$ and $v_2$ be the two vertices in $D$ that correspond to the two faces in $\Pi$ with
$e$ on their boundary;
Compute the shortest path $p$ from $v_1$ to $v_2$ in $D$;
if the cost of $p$ is zero then
    return "true";
else
    return "false";
```

The two algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2` can both be implemented in polynomial time because the size of the SPQR-tree of a planar biconnected graph including all skeletons is linear in the size of the graph [22].

The Embedding Algorithm

Let $\mathcal{S}$ be a skeleton of a P-node. We call the embedding of $\mathcal{S}$ *ν-admissible* if all ν-blue edges for each cluster ν are consecutive and the blue edges that are not ν-permeable (if they exist) are at the beginning and at the end of the sequence.

Our algorithm for finding an augmentation or proving that no augmentation exists works in two phases:

1. Using the colors and attributes of the edges in each skeleton, we fix an embedding for every P- and R-node skeleton and thus determine an embedding for G.
2. We use the algorithm of Section 7.2.1 for fixed embeddings to determine whether an augmentation is possible.

The embedding computed in the first step has the property that it allows an augmentation if and only if there is an embedding of G that allows an augmentation.

We set the embedding for the skeletons of the R- and P-nodes recursively using the structure of the SPQR-tree. We assume that the vertices in G are numbered and that all edges

Algorithm 21: Algorithm `MarkEdgesPhase1` that marks all edges in the skeletons except the virtual edges (Q-nodes are omitted).

```
Input: A node $v$ in $\mathcal{T}'$ and the cluster tree $T$
Result: All edges in $S(v)$ except the virtual are marked.
Let $L$ be the set of children of $v$;
foreach $v' \in L$ do
    MarkEdgesPhase1($v', T$);
    Let $e$ be the edge that $S(v')$ shares with $S(v)$;
    forall the clusters $\nu$ of $T$ do
        Mark all original edges $\nu$-permeable that are either incident or belongs to $\nu$;
        if $S(v')$ contains a $\nu$-blue or $\nu$-permeable edge or a vertex of $\nu$ then
            Mark $e$ $\nu$-blue
        else
            Mark $e$ $\nu$-black;
    switch type of node $v'$ do
        case $P$-node
            forall the clusters $\nu$ of $T$ do
                if all edges in $S(v')$ except $e$ are $\nu$-permeable or one of the vertices
                in $S(v')$ belongs to $\nu$ then
                    Mark $e$ $\nu$-permeable
        case $S$-node
            Take a copy $S'(v')$ of skeleton $S(v')$;
            forall the cluster $\nu$ do
                forall the edges $e_b$ marked $\nu$-blue do
                    if $S'(v')$ contains an edge $e_p$ marked $\nu$-permeable then
                        forall the edges $e_b$ marked $\nu$-blue do
                            Connect source of $e_p$ with source of $e_b$;
            if $S'(v')$ is not planar then
                return error "$C$ not $c$-planar";
            Calculate SPQR-tree of $S'(v')$ and take the skeleton $S''$ that contains the
            same reference edge as $S(v')$;
            forall the cluster $\nu$ do
                if $S''$ contains a vertex of $\nu$ or an edge marked $\nu$-permeable then
                    Mark $e$ $\nu$-permeable
        case $R$-node
            Let $e$ be the edge shared by $S(v')$ and $S(v)$;
            Solve corresponding 2-SAT instance, do corresponding augmentation;
            if findPath($e, v', \nu$) returns true then
                Mark $e$ $\nu$-permeable;
```

Algorithm 22: Algorithm `MarkEdgesPhase2` marks all the virtual edges of the skeletons in the subtree rooted at v if all the edges in the skeleton of v are marked (Q-nodes are omitted)

```
Input: A node $v$ in the SPQR-tree $\mathcal{T}$ where all edges are marked
Result: All edges in the subtree rooted at $v$ are marked
Let $\mathcal{S}$ be the skeleton of $v$;
Let $L$ be the list of edges in $\mathcal{S}$ whose twin edge is contained in a skeleton of a child of $v$;
forall the clusters $\nu$ do
    if $\mathcal{S}$ does not contain a $\nu$-blue or $\nu$-permeable edge or a vertex of $\nu$ then
        Mark all the twin edges of the edges in $L$ $\nu$-black;
    else
        Mark all the twin edges of the edges in $L$ $\nu$-blue;
switch type of $v$ do
    case $S$-node
        Take a copy $S'(v)$ of skeleton $S(v)$;
        forall the cluster $\nu$ do
            forall the edges $e_b$ marked $\nu$-blue do
                if $S'(v')$ contains an edge $e_p$ marked $\nu$-permeable then
                    forall the edges $e_b$ marked $\nu$-blue do
                        Connect source of $e_p$ with source of $e_b$;
                    if $S'(v')$ is not planar then
                        return error "$C$ not $c$-planar";
        Calculate SPQR-tree of $S'(v')$ and take the skeleton $S''$ that contains the same reference edge as $S(v')$;
        if $S''$ contains a vertex of $\nu$ or an edge marked $\nu$-permeable then
            Mark every twin edge of the edges in $L$ $\nu$-permeable
    case $P$-node
        forall the clusters $\nu$ do
            if all edges in $\mathcal{S}$ are marked $\nu$-permeable then
                Mark all twin edges of the edges in $L$ $\nu$-permeable
    case $R$-node
        forall the clusters $\nu$ do
            foreach edge $e$ in $L$ do
                Let $v'$ be the child of $v$ that contains $e$;
                if findPath$(e, v')$ returns $\nu$-permeable then
                    Mark $e$ in $S(v')$ as $\nu$-permeable;
forall the children $w$ of $v$ do
    MarkEdgesPhase2$(w, T)$;
```

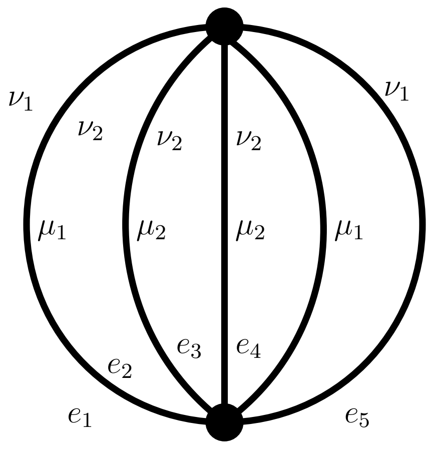


Figure 7.40: An example for the consecutive ones property in a P-node: index 1 is equal to blue, and index 2 means permeable, e.g. ν_1 is therefore equal to ν-blue

are directed from the vertex with lower number to the vertex with higher number.

For simplicity, we consider whether a special case is presented where a planar c-connectivity cannot exist.

Obviously, if we have at least three ν-blue edges in $\mathcal{S}$ for a cluster ν C is not c-planar.

Therefore, we assume that we have at the most two ν-blue edges in $\mathcal{S}$ for each cluster ν.

First, we test whether the biconnected graph contains only P-nodes with skeletons that have at the most two edges. Then we can sort the two ν-blue edges as previous mentioned to obtain an ν-admissible embedding.

Since we are interested in obtaining a planar embedding where we have a ν-admissible ordering for all clusters ν we have to check whether for each P-node its edges has a cyclic consecutive ones property in respect to the ν-blue and ν-permeable attributes for all clusters ν. Furthermore, this cyclic consecutive ones property has to guarantee a ν-admissible ordering for all clusters ν.

In Figure 7.40 an example is given for a P-node.

This figure already shows a ν- and μ-admissible ordering of the edges of the P-node. Note that the ordering of the 1 entries may be cyclic that means that additionally the last entry in the row may be consecutive to the first entry in the same row. This property can be checked using a specific 0/1-matrix in linear time and can be modelled by the so-called data structure PQR-tree [2]. In our situation, the corresponding consecutive ones property can be observed in a corresponding matrix that has a column for each edge of the P-node. The corresponding matrix M of the example in Figure 7.40 is visualized in the following.

$$\begin{array}{c|ccccc|}
 & e_1 & e_2 & e_3 & e_4 & e_5 \\
1.(\nu) & 1 & 1 & 1 & 1 & 1 \\
2.(\nu) & 0 & 1 & 1 & 1 & 0 \\
3.(\nu) & 1 & 1 & 1 & 1 & 0 \\
4.(\nu) & 0 & 1 & 1 & 1 & 1 \\
5.(\mu) & 1 & 1 & 1 & 1 & 0 \\
6.(\mu) & 0 & 1 & 1 & 0 & 0 \\
7.(\mu) & 1 & 1 & 1 & 0 & 0 \\
8.(\mu) & 0 & 1 & 1 & 1 & 0
\end{array} =: M$$

The first four rows of M are dedicated to cluster ν. In the first row every edge gets a 1 if it is ν-blue or ν-permeable. This means that all edges that are ν-blue or ν-permeable should be ordered consecutively. The second row indicates that all edges marked ν-permeable should be ordered consecutive. The third and fourth rows indicate that the two ν-blue edges e_1, e_5 should be in the beginning or end of the sequence. The fifth to last rows are dedicated to cluster μ and build in the same way as for ν.

In general, we have to take a specific case into account that we already discussed in Figure 7.38.

To do this, we fix the skeleton of each P-node temporarily. Since the clustered graph is co-connected, the ν-permeability for a cluster ν occurs in the corresponding pertinent graph in one specific connected region. Consequently, for each virtual edge that has the attribute permeable for certain clusters there exists a unique ordering of those connected regions within the corresponding pertinent graphs. Hence, we can sort the virtual edges by assigning them into their corresponding groups according the ordering and then we code them into a 0/1-matrix to solve the corresponding cyclic consecutive ones problem.

To make this more obvious, we give an example; see Figure 7.41.

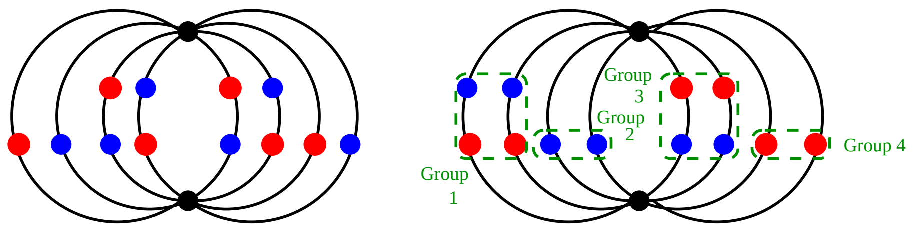


Figure 7.41: A co-connected clustered graph with a feasible embedding on the left-hand side.

In Figure 7.41 we have a co-connected planar clustered graph with two different planar embeddings. In its corresponding SPQR-tree, we have a P-node whose skeleton corresponds to the graph visualized in Figure 7.42. In Figure 7.42, the planar embedding of the skeleton is fixed according to the planar embedding presented on the left-hand side of Figure 7.41.

The co-connected planar clustered graph has two sibling clusters, say blue and red, respectively, that contain blue and red colored vertices, respectively. We recognize that the planar embedding of the clustered graph visualized on the right-hand side of Figure 7.41 can be augmented to a c-connected planar clustered graph since the planar embedding of the subgraphs connecting the split pairs is sorted by groups of the same properties. Obviously, this planar embedding is c-planar while the embedding on the left-hand side of Figure 7.41 is not. Additionally, those groups depend on the ordering of the pairwise disjoint cluster pair blue, red which inherit the attributes blue- and red-permeable to the corresponding virtual edges in the SPQR-tree.

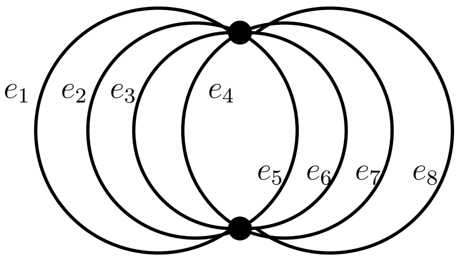


Figure 7.42: The corresponding skeleton of the P-node that is contained in the SPQR-tree of the clustered graph visualized in Figure 7.41 - the planar embedding of the skeleton corresponds to the planar embedding on the left-hand side of Figure 7.41

In Figure 7.42, virtual edges e_1 to e_6 get attributes blue-permeable, and virtual edges e_3 to e_8 get attribute red-permeable according to the coloring algorithm presented in the previous section. Neverless, according to the above mentioned ordering, the groups are classified as follows: virtual edges e_i, e_{i+1} with $i = 1, 3, 5, 7$ belong to group $(i+1)/2$. When we take a closer look we recognize that group 1 and 3, respectively, have to occur in between group 2 and 4 in a feasible planar embedding. Clearly, group 1 and 3 are not allowed to be consecutive in a feasible planar embedding.

Next the question occurs how to determine those groups. We already observed that the ν-permeability for a cluster ν occurs in the corresponding pertinent graph in one specific connected region. Consequently, if we fix the planar embedding of the corresponding skeleton that belongs to a P-node, see Figure 7.42, we traverse the virtual edges according to their ordering in the fixed planar embedding, e.g. in our example from e_1 to e_8. We already observed that in each virtual edge with the attribute permeable for at least two clusters those clusters are ordered hierarchically in the corresponding pertinent graph. Therefore, there is a unique hierarchical ordering of those clusters in the pertinent graph of the corresponding virtual edge. Hence, we check this ordering for any such pair of clusters in each virtual edge. If a pair of those clusters occur the first time, we assign to their ordering $+1$. For example, see Figure 7.42, for the virtual edge e_3, the vertex of the red cluster is ordered before the vertex of the blue cluster, say the ordering is red/blue. For this ordering we assign $+1$. If there exists already an ordering with $+1$ and this pair of clusters occurs in an other virtual edge with the reversed ordering, then we assign to this virtual edge for this pair of clusters −1. In Figure 7.41 and 7.42 for virtual edge e_4, the vertex of the blue cluster is ordered before the vertex of the red cluster, say the ordering is blue/red. For this ordering we assign −1. Finally, all virtual edges with a red/blue ordering +1 are assigned to group 3, whether the virtual edges with a blue/red ordering −1 are assigned to group 1. Virtual edges, that have only one of the two clusters and are marked permeable for that cluster, are assigned to group 2 or 3, respectively.

We transform this restrictions into a 0/1-matrix M in the following way:

Firstly, all vertices that belong to the blue and red cluster, respectively, have to be consecutive in a feasible planar embedding. This fact forms the first two rows of M. Additionally, the vertices that are contained in one of the groups 1 to 4 have to be consecutive in a feasible planar embedding; this is coded in the rows 3 to 6 in M. Furthermore, group 1 and 3, respectively, has to be sorted in between group 2 and 4; see rows 7, 8 and 9, 10, respectively, of M.

$$M = \begin{array}{l} \\ 1.(blue) \\ 2.(red) \\ 3.(group1) \\ 4.(group2) \\ 5.(group3) \\ 6.(group4) \\ 7.(group1\&group2) \\ 8.(group1\&group4) \\ 9.(group2\&group3) \\ 10.(group3\&group4) \end{array} \begin{array}{c} \begin{array}{cccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Observe that for each virtual edge we have to take into account any pair of clusters ν, μ with attributes ν-, μ-blue in the corresponding pertinent graph in which at least one has additional the attribute ν- or μ-permeable, respectively. All ν-blue marked virtual edges for a cluster ν have to be ordered in the same manner as we described for ν-permeable marked virtual edges. In Figure 7.43 the vertex v has to be embedded consecutively to group 1 which results in a feasible planar embedding. Embedding v consecutively to group 2 would result in a not feasible planar embedding.

We build such a matrix M for each P-node and test the cyclic consecutive ones property of its 1 entries. If no one exists C cannot be c-planar and we stop. Otherwise we fix the planar embedding of the P-nodes in the order as determined by the cyclic consecutive ones property. In the example of Figure 7.41 and 7.42 a feasible planar embedding is achieved by the ordering e_4, e_6, e_2, e_8, e_3, e_5, e_1, e_7, visualized on the right-hand side of Figure 7.41.

Recall that we have fixed the planar embedding of each skeleton of a P-node. The planar embedding of an S-node is unique hence all the skeletons of S-nodes are fixed by construction. Consequently, we have still to fix the planar embedding of the skeletons of the R-nodes. Recall that by construction a skeleton of an R-node has exactly two planar

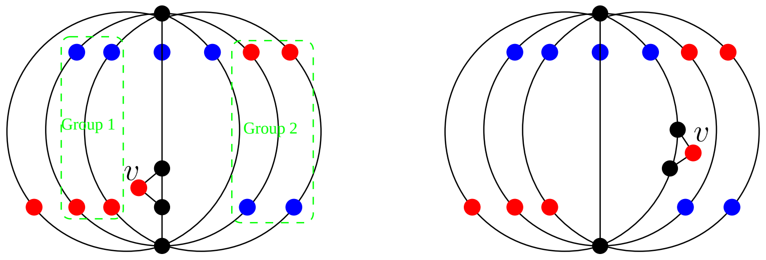


Figure 7.43: Calculation of a feasible planar embedding of P-nodes.

embeddings and we switch from one to the other by swapping.

The calculation that is left to be done is the embedding of the pertinent graphs of all ν-blue edges in the partially fixed planar embedding so far.

Observe that for all ν-permeable edges the embedding is also fixed so far. Consequently, we can already augment all vertices that are involved in the ν-permeability of a virtual edge.

For all other vertices involved in a cluster ν we do the following. For each cluster ν we test whether a pair of its vertices e belong to a face f of the modified graph G. If so, we mark the pair and assign the boolean variable x_f^e to the pair of vertices. We identify e as a potential edge that might be feasible in the final planar embedding. x_f^e gets value 1 if e is feasible and 0 otherwise. Observe that the number of those pairs is $O(n^2)$.

To achieve a fixed planar embedding out of the partially fixed planar embedding we transform the edge-augmentation problem left so far into an other problem, the 2-XOR SAT, that is in $\mathcal{P}$ [75].

The decision problem 2-XOR SAT is given in the following way:

Given: Set U of variables, collection C of clauses with XOR over U such that each clause $c \in C$ has $|c| = 2$.
Question: Is there a satisfying truth assignment for C?

Our aim now is to develop a 2-XOR SAT instance that codes the conflicting clusters for each face. If the instance is feasible it will give us a fixed planar embedding of the partially fixed planar embedding so far such that the faces have no conflicting clusters. Afterwards, we have to check whether or not the modified clustered graph with the constructed planar embedding is c-connected. In the positive case the original input clustered graph is c-planar.

To do so, we need to consider pairwise disjoint pairs of vertices that belong to different non-descendant clusters. Additionally, this pair of vertices has to be embeddable into the same face. Observe that then they form a potential connection in a cluster in order to achieve c-connectivity of the clustered graph. Recall that there is at most one of such a face for a co-connected clustered graph. This face can be determined by a simply applying the embedding step of SIPCA; see Chapter 6. Furthermore, recall that each planar embedding can be achieved by fixing the planar embedding of each skeleton of the SPQR-tree. Since we have already fixed the planar embedding of each P-node, each face of the partially fixed embedding corresponds to exactly one face in exactly one skeleton of the SPQR-tree. This face is uniquely defined by its split pairs that surrounds it. Observe that those faces never contain the reference edge of a skeleton in its boundary. In the following we identify the face with the corresponding face in a skeleton of the SPQR-tree since we are dealing with a partially fixed planar embedding so far. We declare a list for each face, and a list for each vertex of the pairs. In the first step we check each pair of vertices v, w if they satisfy the above properties. If they do, we assign a boolean variable $x^f_{(v,w)}$ to them, (v, w) is regarded as a potential edge, mark them feasible and save the variable in the lists of the corresponding face f and of the corresponding end vertices v, w.

Recall that we have $O(n^2)$ such pairs, hence we have $O(n^4)$ pairwise disjoint pairs. Whether two clusters are non-descandant can be determined very easily: this is the case if and only if the lowest common ancestor of two clusters is disjoint from them.

Recall that we have a list in which we saved all boolean variables that belong to the same face f. We consider pairwise the pairs of vertices in it. Let e_1 and e_2 be such a pair which belong to the same face f. If they are conflicting we can embed *either* e_1 *or* e_2 in f. Let $x^f_{e_1}$ and $x^f_{e_2}$ be the boolean variables of two conflicting pairs e_1 and e_2. We add the following clause to the 2-XOR SAT instance; see Figure 7.44:

$$x^f_{e_1} \dot{\vee} x^f_{e_2} = (\neg x^f_{e_1} \vee \neg x^f_{e_2}) \wedge (x^f_{e_1} \vee x^f_{e_2}) = (\neg x^f_{e_1} \wedge x^f_{e_2}) \vee (x^f_{e_1} \wedge \neg x^f_{e_2}) = x^f_{e_1} \oplus x^f_{e_2}$$

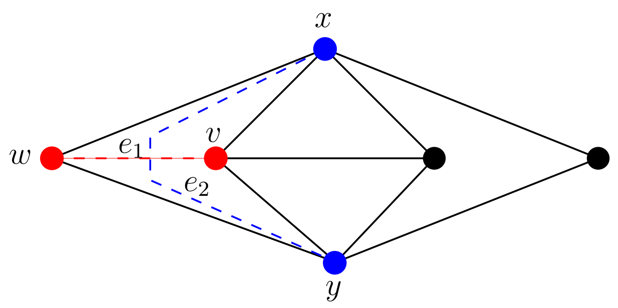


Figure 7.44: Transformation in 2-XOR SAT clause - first case

Observe that those conflicts can be determined easily in a face since the ordering of the corresponding vertices of each pair is alternating; see Figure 7.44.

Additionally, if we have two pairs of vertices $e_1 = (v, w)$, $e_2 = (v, z)$ that contain each the same vertex v that can be embedded *either* in f *or* in g, we add $x^f_{e_1} \oplus x^g_{e_2}$ to the 2-XOR SAT instance; see Figure 7.45. Recall that we have a list for vertex v that contains the corresponding boolean variables for the two faces f and g.

Finally, we have the last case visualized in Figure 7.46. There we see that v and w cannot be embedded into the same face. Consequently, if e_1 is the pair of vertices that contains v

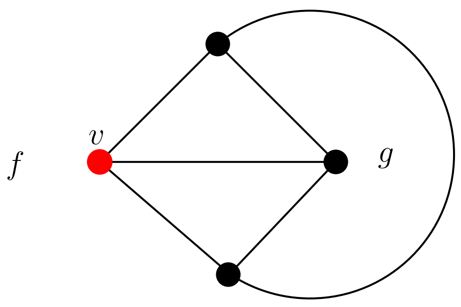


Figure 7.45: Transformation in 2-XOR SAT clause - second case

and e_2 is the pair of vertices that contains w we have $x^f_{e_1} \oplus x^f_{e_2}$ and $x^g_{e_1} \oplus x^g_{e_2}$.

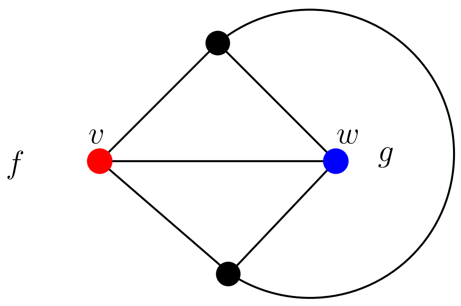


Figure 7.46: Transformation in 2-XOR SAT clause - third case

Recall that the planar embedding of the skeletons of all R-nodes are left to be fixed. To do so we traverse the SPQR-tree from bottom to top level by level. For each R-node on the way we add the previous mentioned 2-XOR SAT clauses for all virtual edges that are ν-blue but not ν-permeable. This can be done and inherited easily. Once we arrive at the root of the SPQR-tree we have completed our 2-XOR SAT instance.

This 2-XOR SAT instance can be solved easily: Recall that in each clause once we fix one variable the other is also fixed. Additionally, all other variables joining those variables in other clauses are fixed. Obviously, we have some sort of connected component induced by all those variables. Consequently, we have to solve all those connected components by assigning either 1 or 0 to one of its variables each. Once a contradiction in assigning of boolean values occurs we know that the original 2-XOR SAT instance is not feasible. Otherwise, we have a feasible solution.

Once we have a feasible solution of the 2-XOR SAT instance this can be transformed to a fixed planar embedding of the original clustered graph out of the partially fixed planar embedding so far. Observe that each pair of vertices e can be connected by an edge if the corresponding boolean variable in the feasible 2-XOR SAT solution is set to 1. We only need to add those edges to the partially fixed planar embedding so far. The only thing left to be done is now to check whether or not this planar embedding induces a c-connected planar clustered graph. We do this by running the algorithm for the triconnected case on the calculated planar embedding. In the positive case the original clustered graph has a planar c-connected super-clustered graph and therefore is c-planar; otherwise it is not c-planar.

7.2.3 The Algorithm for the Connected Case

We assume that G is connected but not biconnected. Furthermore, for simplicity we first suppose that G is a tree.

Assume that we have two vertices v_1 and v_2 of a cluster ν. Furthermore, we assume that v_1 and v_2 belong to different connected components of $G(\nu)$ (see Figure 7.47).

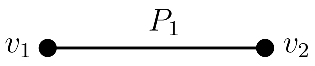


Figure 7.47: Tree case: connectivity considerations

Since G is connected there does not exist two disjoint paths but one path from v_1 to v_2, say P_1 (see Figure 7.48). Further, since v_1 and v_2 are not connected in $G(\nu)$ P_1 use vertices of $G - G(\nu)$. Since $G - G(\nu)$ is connected and G is a tree (and therefore no other path other than P_3 exists that might connect connected components of $G - G(\nu)$) all vertices of $G - G(\nu)$ have to be contained in P_1 and connected to each other.

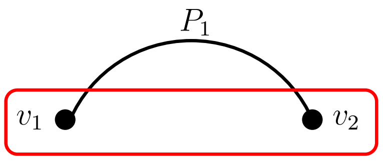


Figure 7.48: Tree case: connectivity considerations

Consequently, v_1 and v_2 are leaves of G or are each connected to a leaf in $G(\nu)$.

Since for each cluster ν $G - G(\nu)$ is connected ν might have several connected components C_i for $i \in \mathbb{N}$. Then C_i is equal to a subtree G_i in G by induction. Additionally, G_i contains all vertices that are descendants of the lowest common ancestor of C_i in G. Figure 7.49 gives an example.

Observe, once the leaves of two subtrees of G contained in a cluster ν are ordered consecutive the subtrees are ordered consecutive. Therefore, we may restrict ourselves on ordering the leaves in respect to the inclusion relation of the clusters they belong to.

One way to do so is to take G as a PQ-tree and the leaves of each cluster for one restriction set.

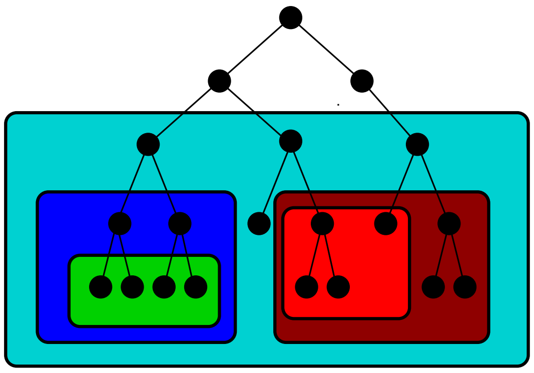


Figure 7.49: A co-connected clustered graph $C = (G, T)$ with a tree G

The other more efficient way is to use the following theorem.

Theorem 7.9. *Let $C = (G, T)$ be a co-connected clustered graph and G a tree. C is c-planar if and only if $G \cup T$ is planar.*

Proof.

1. We assume that $G \cup T$ is planar. Since T is the clustertree of C, $C' = (G \cup T, T)$ is c-connected. Since, C' is a super-clustered graph of C it is additionally co-connected. Consequently, C' is c-planar. Since C is a sub-clustered graph of C' C is also c-planar.

2. We assume that C is c-planar. Since C is co-connected and G a tree there exists a consecutive ordering of the leaves that belong to cluster ν for each cluster ν in T. Consequently, if we assume that each such leaf is a representive of ν in the beginning, we may add for each cluster ν a star connecting its representives. In each step we reassign the representives such that an added star vertex v becomes a representive of ν instead of the leaves that are connected with v. Clearly, the union of stars is a tree that is isomorphic to the cluster tree T. Therefore, we have a clustered graph $C' = (G \cup T, T)$ that is still c-planar. Then, $G \cup T$ is planar.

□

Since G is planar and hence C has a linear number of (trivial and non-trivial) clusters, $G \cup T$ can be tested in linear time for planarity. Consequently, C can be tested for c-planarity in linear time by Theorem 7.9.

In the following we assume that G is planar and connected but not biconnected.

In the last section we have dealt with the problem restricted on biconnected graphs. Hence we already know how to deal with the biconnected blocks of the graph G using the SPQR-tree. To solve the connected case, we first build the BC-tree of G. Recall that this tree has two types of nodes: The c-nodes correspond to cut vertices of G and the b-nodes to biconnected components (blocks). There is an edge connecting a c-node and a b-node, if the cut vertex is contained in the block corresponding to the b-node. Taking this structure into account we can solve the problem also for connected graphs in polynomial time.

We split the connected graph G into biconnected components using the BC-tree and we split the biconnected blocks into triconnected components using the $SPQR$-tree. Then we connect the blocks by augmenting the vertices of a cluster ν using the cut vertices in bc.

It is obvious that we have to operate only on the smallest subtree bc of the BC-tree that contains all clusters ν of T.

For each cluster ν, we mark a cut vertex ν-red that is contained in a block with a vertex of ν.

Further, for each cluster ν of T we add a dummy vertex and connect it to cut vertices that are marked ν-red.

Then we test planarity of the modified BC-tree.

The ν-blue blocks can be connected in the same face if and only if the planarity testing results true. In the positive case, we mark all cut vertices that are ν-red and belong to ν ν-blue. For each ν-blue biconnected component its ν-red vertices play a special role since they represents the connection to the other ν-blue biconnected component in bc. Consequently, its ν-blue vertices has to be connected with the ν-red vertices to ensure a planar connectivity augmentation for all ν-blue vertices. Hence, for each biconnected component that is marked ν-blue we apply the algorithm of the biconnected case presented in the last section on its ν-blue and ν-red vertices. In the coloring algorithms, Algorithms 21 and 22, we differ between ν-blue and ν-red vertices: clearly, the ν-red vertices has to appear in the beginning or the end of the sequences. Consequently, we assign the attribute ν`-blue` to a virtual edge e if e has no attribute ν`-permeable` created by the ν-blue vertices of its expansion graph but its expansion graph contains a ν-red vertex. We apply for all biconnected blocks B the algorithm `CheckAugmentability` of the biconnected case and extend it for the ν-red cut vertices as described.

7.2.4 The Algorithm for the General Case

We assume that G is not connected. Since $G-G(\nu)$ is connected for each cluster ν, $G-G(\nu)$ is contained in exactly one connected component of G for each cluster ν. Consequently, every vertex of ν connected to $G-G(\nu)$ is contained in the same connected component as $G-G(\nu)$. An example is given in Figure 7.50.

Obviously, if there exists a component of G in which vertices of a cluster ν are contained but no vertices of $G-G(\nu)$ then this connected component can be placed in the final planar embedding into a face that itself contains at least one vertex of ν. Consequently, we have to take care about all connected components that contain vertices of $G(\nu)$ *and* of $G-G(\nu)$ for a cluster ν.

For those connected components, we apply a modified variant of the algorithm for the

Algorithm 23: Algorithm `ExtendedCheckAugmentability` checks if a planar c-connectivity augmentation for T in a connected graph G exists.

```
Input: A connected planar graph.
Result: An embedding of $G'$ if $G$ can be augmented, otherwise "false".
Calculate $BC$-tree of graph $G$;
forall the blocks $B$ of $BC$-tree do
    forall the clusters $\nu$ of $T$ do
        if $B$ contain a vertex of $\nu$ then
            Mark $B$ $\nu$-blue;
Calculate the smallest $BC$-subtree $bc$ containing all $\nu$-blue blocks;
forall the $C$-nodes $c$ in $bc$ do
    forall the clusters $\nu$ of $T$ do
        if $C$-node $c$ is not $\nu$-blue then
            mark $c$ $\nu$-red;
forall the clusters $\nu$ of $T$ do
    forall the blocks $B$ of $bc$ which are not $\nu$-blue do
        Join all $\nu$-red nodes of the blocks $B$ with a single dummy vertex that represents $\nu$;
Test planarity;
if not planar then
    return error "$C$ not $c$-planar";
forall the clusters $\nu$ do
    forall the blocks which are $\nu$-blue do
        Apply changed algorithm CheckAugmentability with $\nu$-red nodes;
```

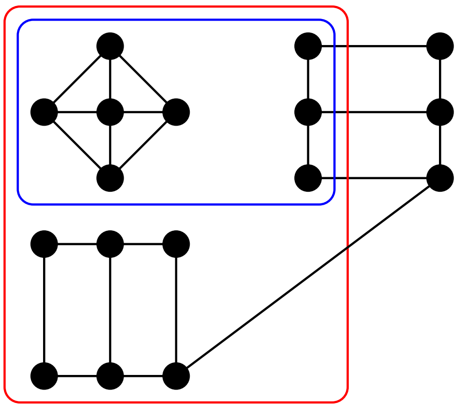


Figure 7.50: An example of a co-connected planar clustered graph $C = (G,T)$ with a disconnected underlying graph G.

connected case of the previous section (see Algorithm 24) to each of those connected component. The difference is that we transform those connected components so that they have at least one ν-blue and all ν-red vertices on the outer face for each cluster ν. Finally, for each cluster ν, bottom up level by level in the cluster-tree of C, we connect the connected components by connecting its ν-blue vertices in the outer face.

Theorem 7.10. *Let a co-connected planar clustered graph $C = (G,T)$ be given. c-Planarity can be tested and in the positive case a c-planar embedding can be achieved in polynomial time.*

The remaining results of this chapter are joint work with Carsten Gutwenger, Michael Jünger, Sebastian Leipert, Petra Mutzel and René Weiskircher and published in [40].

7.3 Clustered Graphs with Two Clusters

Let $C = (G,T)$ be a clustered graph with a root cluster and exactly one additional cluster ν. Let the graph G be connected and the subgraph induced by the vertices of the cluster ν non-connected. Observe that $G - G(\nu)$ might also be disconnected. The problem of connecting the subgraph induced by one cluster is similar to the problem of planar connectivity augmentation of an induced subgraph discussed in Chapter 6.

In the following, recall from Chapter 6, we name the vertices of $G(\nu)$ *blue vertices*. After constructing an SPQR-tree $\mathcal{T}$ for every biconnected component of G we mark for every SPQR-tree each edge in every skeleton either blue or black. Recall that an edge of a skeleton is marked blue, if its expansion graph contains blue vertices. Otherwise it is marked black.

Additionally, recall from Chapter 6, we assign an attribute called *permeable* to certain blue edges. Intuitively, an edge is permeable if it is possible to construct a path connecting only

Algorithm 24: Algorithm `GeneralCheckAugmentability` checks if a planar connectivity augmentation for ν in a general graph G exists.

```
Input: A general co-connected clustered planar graph $C = (G, T)$.
Result: An embedding of $C$ that can be $c$-connectivity augmented, otherwise
        "false".
forall the connected components $C_i$ of the graph $G$ do
    ExtendedCheckAugmentability;
Sort all $C_i$ that contains $G(\nu)$ and $G - G(\nu)$ for a cluster $\nu$ in clockwise order and
traverse them in this order in the following;
forall the $C_i$ which contains at least one $\nu$-blue or $\nu$-red vertex do
    Choose a face $f$ with at least one $\nu$-blue or $\nu$-red vertex for each cluster $\nu$ in the
    boundary as outer face;
    forall the clusters $\nu$ of $T$ do
        Save the $\nu$-blue or $\nu$-red vertex in a list listvertex($\nu$);
    forall the clusters $\nu$ of $T$ do
        forall the vertices in listvertex($\nu$) do
            Connect them by inserting clockwise a path;
```

blue vertices through its expansion graph. Let $G(e)$ be the expansion graph of edge e in skeleton $\mathcal{S}$. Recall that since $G(e)$ is biconnected we have that in any planar embedding $G(e)$ there are exactly two faces that have e on their boundary. Recall that the edge e in $\mathcal{S}$ is permeable with respect to W, if there is an embedding Π of $G(e)$ and a list of at least two faces $L = (f_1, \ldots, f_k)$ in Π that satisfies the following properties:

1. The two faces f_1 and f_k are the two faces with e on their boundary.

2. For any two faces f_i, f_{i+1} with $1 \leq i < k$, there is a blue vertex on the boundary between f_i and f_{i+1}.

Recall from Chapter 6 that we call a skeleton $\mathcal{S}$ of a node $\wp$ of $\mathcal{T}$ permeable if the pertinent graph of $\wp$ and the virtual edge of $\mathcal{S}$ have the two properties stated above. Recall that $\mathcal{S}$ is permeable if the twin edge of its virtual edge is permeable.

Theorem 7.11. *Let $C = (G, T)$ be a clustered graph such that the following conditions hold:*

- *G is series-parallel*

- *C contains only one non-trivial non-root cluster ν and W is its corresponding vertex set.*

Let $\mathcal{T}$ be the set of the SPQR-trees of every biconnected component of G. Let $\mathcal{C} := \{\mathcal{V} \in 2^W | \mathcal{V}$ is a circle in the expansion graph of a P-node and contains both pole vertices$\}$ be the set of the vertex sets of all circles in the expansion graphs of P-nodes that contain both pole vertices. Let the subgraph G_W induced by W in G allow a planar connectivity augmentation for W. If for every P-node the following property $(*)$ holds, then C is c-planar:

$(*)$ *In every P-node, there is at the most one circle $\mathcal{V}$ of $\mathcal{C}$ such that*

1. *the union of the expansion graphs of two children $\wp_1$ and $\wp_2$ contains $\mathcal{V}$ and*
2. *for $i \in \{1, 2\}$ the cut of $G - G_W$ with the expansion graph of child $\wp_i$ is nonempty.*

Proof. We calculate the BC-tree of G and for every block B the $SPQR$-tree of its biconnected component. The $SPQR$-tree of a series-parallel graph does not contain R-nodes.

Hence G_W has a planar connectivity augmentation, we assume that G_W is connected. As we introduce a minimum cardinality edge set applying the planar connectivity augmentation, we do not loose c-planarity. According to Theorem 2.3 we need to show that a planar embedding exists such that the subgraph $G - G_W$ is embedded into the outside of G_W. It follows that we need to show that $G - G_W$ is not embedded partially inside of G_W (if it is embedded completely in an inner face we choose a face f that has an edge $e = (v, w)$ with $v \in W$ and $w \notin W$ as outer face). Consider now the P-, S- and Q-nodes. For P-nodes we have the following cases

- the P-node is black, means it contains only black edges,
- the P-node is blue and there exists exactly one blue edge in a P-node. The expansion graph of this blue edge contains the subgraph G_W or
- the P-node is blue and permeable.

A blue P-node that is not permeable such that case 2 does not hold cannot exist for the following reasoning. If there is at least one blue edge in the P-node and another blue vertex or blue edge in another node, at least one pole vertex has to be blue due to the connectivity of G_W. If there are at least two blue edges in the P-node, the pole vertex must be blue.

Similar reasoning holds for S- and Q-nodes of the SPQR-trees and for the cut vertices of graph G. The latter are blue if at least two blocks which they belong to are blue.

We have to show that there does not exist a planar embedding such that the subgraph $G - G_W$ is embedded in the outside of G_W if and only if there exists a P-node $\mathcal{P}$ with more than one circle of vertices of W in the corresponding expansion graph fulfilling conditions 1 and 2. Note, that $G(\nu)$ is equal to G_W and that $\mathcal{P}$ is permeable in this case since both pole vertices belong to $G(\nu)$.

If there exists a P-node $\mathcal{P}$ with more than one circle of vertices of W in the corresponding expansion graph fulfilling conditions 1 and 2., then we order the edges that correspond to the union of expansion graphs fulfilling conditions 1 and 2. consecutively in $\mathcal{P}$. We want to find an embedding according to Theorem 2.3. As we have only two clusters (a root cluster and cluster ν) this is equal to the fact, that $G(\nu)$ can be embedded into the outside of $G - G(\nu)$. As $G(\nu)$ is connected (there exists a planar connectivity augmentation) there has to be an embedding Π of G with a sequence of faces $f_1, \dots, f_k$ such that there is at least one vertex of $G - G(\nu)$ in the boundary between two consecutive faces f_i and f_{i+1} and the boundaries of all those faces contain all vertices of $G - G(\nu)$. This is equal to the fact that $G - G_W$ has a planar connectivity augmentation for $V - W$ in G as described in Chapter 6. The faces containing vertices of $G - G(\nu)$ in their boundaries in $\mathcal{P}$ are contained in the sequence of faces $f_1, \dots, f_k$. As the pole vertices are contained in $G(\nu)$ and there exists at least two circles of vertices of $G(\nu)$ that are contained in at least three expansions graphs, the sequence of faces $f_1, \dots, f_k$ cannot be consecutive and therefore there cannot exist a planar connectivity augmentation of $G - G_W$ for $V - W$ in G. Therefore, we cannot find an embedding according to Theorem 2.3 and therefore C is not c-planar.

If C is not c-planar, then there is an embedding Π in which the vertices of $G - G(\nu)$ cannot be embedded into the outside of $G(\nu)$ according to Theorem 2.3. As $G(\nu)$ is connected (there exists a planar connectivity augmentation) there does not exist a sequence of consecutive faces $f_1, \dots, f_k$ which are consecutive so that there is at least one vertex of $V - W$ on the boundary between two faces and all vertices of $V - W$ are included in the union of the boundaries of $f_1, \dots, f_k$. Therefore there exists a sequence of faces $f_1, \dots, f_k$ in which their boundaries contain all vertices of $G - G(\nu)$ and there is a minimum number of consecutive faces f_i and f_{i+1} so that on the boundary between f_i and f_{i+1} is no vertex of $G - G(\nu)$. This is equal to the fact, that on the boundary between f_i and f_{i+1} are vertices of $G(\nu)$. Let $\mathcal{F}$ be the set of all the faces f_i and f_{i+1}. As the S- and Q-nodes have skeletons with only one embedding and a skeleton represents the whole corresponding biconnected component in G, we have to consider the P-nodes. As the pole vertices are contained in $G(\nu)$ and $\mathcal{F}$ is a minimum cardinality face set, there has to be l paths of vertices of $G(\nu)$ from one pole vertex to the other with $l = \frac{|\mathcal{F}|}{2} + 2$ and $l \geq 3$. Combining the paths to circles (the first and the last vertex of the paths are the pole vertices), we get more than one circle fulfilling condition 1 and 2. $\square$

Note, that $(*)$ in the previous theorem can be replaced by: There exists a planar connectivity augmentation for $V - W$ in G, if the subgraph G_W is connected using the planar connectivity augmentation.

Observe that the previous theorem can be easily extended to c-connected clustered graph $C = (G, T)$ with series-parallel underlying graph G, already stated in Theorems 7.3 and 7.4 that we proved in Section 7.1.

Recall from Section 7.1 that c-planarity can be destroyed in the expansion graphs of R-nodes and in the expansion graphs of P-nodes even if the clustered graph is c-connected.

Theorem 7.12. *A connected clustered graph $C = (G, T)$ where G is planar with one non-connected cluster ν is c-planar if and only if*

1. *there exists a planar connectivity augmentation of the subgraph induced by the non-*

connected cluster ν and

2. *there is an embedding of G that contains no circle of vertices of $G(\nu)$ that separates vertices of $G - G(\nu)$.*

Proof. The proof follows from the previous theorems. □

Note, that item 2 in the previous theorem can be replaced by: There exists a planar connectivity augmentation for the vertices of $G - G(\nu)$ in G.

As a result, we are able to deal with c-planarity of a special subclass of clustered graphs using planar connectivity augmentation and SPQR-trees.

According to Theorem 7.12, we know how to test planar connectivity augmentation of a subgraph of a planar graph. We now show how to test whether there exists an embedding of G that contains no circle in $G(\nu)$ that separates $G - G(\nu)$.

Consider a clustered graph $C = (G, T)$ that has only one non-root cluster ν that is non-connected. Therefore the subgraph $G(\nu)$ has more than one connected component. As we choose a minimum cardinality planar augmenting edge set $\mathcal{M}$ and take the pole vertices belonging to $G(\nu)$ into account to augment C to a c-connected clustered graph and if such an augmentation exists, c-planarity of C is maintained if C is c-planar. Therefore to test whether C is c-planar is equal to the following:

Let C_i, $i = 1, \ldots, l$, $l \geq 2$ be the connected components of $G(\nu)$.

I. There exists a drawing such that the drawing of $G - C_i$ can be drawn outside of the drawing C_i for all $i = 1, \ldots, l$ and

II. there exists a planar connectivity augmentation between all C_i, $i \in \mathbb{N}$ and therefore for $G(\nu)$.

We test I. on an auxiliary modified clustered graph $\tilde{C}$ of C as follows. Cluster ν is split into l dummy clusters so that each connected component C_i $i = 1, \ldots, l$ corresponds to a dummy cluster. By construction $\tilde{C}$ is a c-connected clustered graph and can be tested according Feng, Eades and Cohen [29, 30] and according Dahlhaus [18, 20] for c-planarity.

Definition 7.2. *Let C be a clustered graph with a connected root cluster and a non-connected cluster ν. Let C_{sub} be the clustered graph created by splitting ν into one dummy cluster for each connected component of the subgraph induced by ν that contains at least two vertices (see Fig. 7.51). A connected component that contains only one vertex is treated as a trivial cluster. We call C_{sub} the c-split clustered graph of C.*

Note, that C_{sub} is a c-connected clustered graph.

Theorem 7.13. *Let C be a clustered graph with a connected root cluster and a non-connected cluster ν. Let C_{sub} be its c-split clustered graph. If C_{sub} is not c-planar then C is not c-planar.*

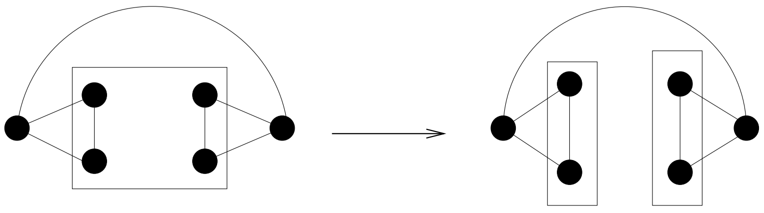


Figure 7.51: A clustered graph C with a non-connected cluster. By splitting the non-connected cluster, C is extended to a c-connected clustered graph.

Proof. If C_{sub} is not c-planar, there exists an edge crossing or a cluster crossing in at least one cluster μ. The subgraphs induced by the dummy clusters of C_{sub} are connected components of the corresponding non-connected cluster ν of C. Thus there exists an edge crossing or a cluster crossing in ν and C is not c-planar. □

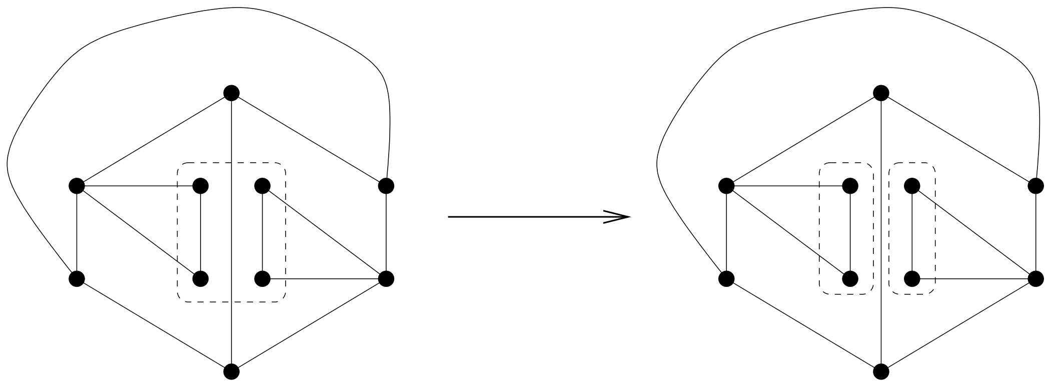


Figure 7.52: An example where C is not c-planar but its c-split clustered graph C_{sub} is c-planar

Note that C must not be c-planar if its c-split clustered graph C_{sub} is c-planar (see Fig. 7.52). In the case that C_{sub} is c-planar and C is not there will not exist a planar connectivity augmentation in C for the non-connected cluster ν.

Thus after a positive result and after the application of the c-planarity test by Cohen, Feng, Eades, [29, 30, 18, 20] resp. Dahlhaus [29, 30, 18, 20] we apply the planar connectivity augmentation algorithm on G and the vertices of ν as subset W (if we have pole vertices that belong to $G(\nu)$, we use them for connectivity). Together with Theorem 7.12 we get the following theorem.

Theorem 7.14. *Let $C = (G,T)$ be a connected clustered graph and ν its only non-trivial non-root cluster that is non-connected. C can be tested for c-planarity in linear time with respect to the number of vertices of C and in the positive case embedded in linear time.*

Algorithm 25: The algorithm for clustered graphs $C = (G, T)$ that contain a connected root cluster and a non-connected cluster ν. It computes an embedding Π and the minimum cardinality augmenting edge set.

```
Input: A clustered graph $C = (G, T)$ that contains a connected root cluster and a
       non-connected cluster $\nu$.
Result: true if and only if there is a $c$-planar connectivity augmentation for $\nu$; in
        the positive case an embedding $\Pi$ and the minimum cardinality augmenting
        edge set.
Compute $C_{sub}$ by splitting the non-connected cluster for each connected component
of the subgraph $G(\nu)$;
Apply the linear time $c$-planarity test on $C_{sub}$;
if the test return false then
    return false;
Apply the subgraph induced planarity augmentation algorithm for $C$ and $\nu$;
if a planar connectivity augmentation exists then
    Compute $\Pi$;
    return true;
else
    return false;
```

Next we consider the case that G is non-connected and there is only one cluster ν that is not the root cluster and is non-connected. For all connected components of G we apply our algorithm for clustered graphs with one non-connected cluster. Then we choose for each connected component a face as outer face that contains at least one blue vertex v_1 and one non-blue vertex v_2. We connect the blue vertices in the outer face so that the edges of a minimum cardinality augmenting edge set is inserted (as described in Chapter 6). In the positive case a c-planar embedding with a minimum cardinality augmenting edge set will be computed. Thus we have that the clustered graph has a c-planar connectivity augmentation for the non-connected cluster ν which leads us to the following theorem.

Theorem 7.15. *Let $C = (G, T)$ be a not necessarily connected clustered graph and ν its only non-trivial non-root cluster that is non-connected. C can be tested for c-planarity in linear time concerning the number of vertices of C and in the positive case embedded in $O(n)$ time.*

7.4 One-Level Clustered Graphs

As shown in the last section it is possible to test c-planarity of a clustered graph with one non-connected cluster and a root cluster in $O(n)$ time, where n is the number of vertices of G. Next we consider a clustered graph $C = (G, T)$ with a cluster tree T with only one level below the root cluster. So now we allow more than one non-root cluster. Further let only one child cluster of the root cluster be non-connected. We assume, that every non-trivial cluster has at least two vertices.

We construct a c-split clustered graph C_{sub} of C as described in Section 7.3. If G is not connected, we apply this technique to every connected component of G.

First, we test if the c-split clustered graph C_{sub} of C is c-planar. We call G_{mod} by the c-planarity test of Feng, Eades and Cohen [29, 30] modified graph G. Then we apply the planar connectivity augmentation algorithm described in Chapter 6 for the vertices belonging to ν in G_{mod}. If one exists, then an embedding Π and a minimum cardinality augmenting edge set is computed and `true` is returned. Otherwise, C is not c-planar. If we get Π, we can apply the techniques used in the c-planarity embedding algorithm of Feng, Eades for the connected clusters using Π to obtain a c-planar embedding of C. For the case that the root cluster is non-connected, we apply further the planar connectivity augmentation algorithm for the root cluster.

Theorem 7.16. *Let $C = (G, T)$ be a connected clustered graph with a cluster tree T with only one level below the root cluster and ν its only non-root non-connected cluster. Let C_{sub} be the c-split clustered graph of C. Let G_{mod} be the modified graph G obtained by the c-planarity test by Feng, Eades and Cohen [29, 30] of $C_{sub} = (G, T_{sub})$. C is c-planar if and only if C_{sub} is c-planar and there exists a planar connectivity augmentation of the vertices belonging to the subgraph $G(\nu)$ in G_{mod}.*

Proof.

- "$\Leftarrow$" We have that C_{sub} is c-planar and there exists a planar connectivity augmentation of $G(\nu)$ in G_{mod}. Thus G is planar and there exists a planar drawing of G such that for every node μ of T_{sub}, all the vertices and edges of $G - G(\mu)$ are in the external face of the drawing of $G(\mu)$. Note that this holds for the connected clusters of C and for the dummy clusters in C_{sub} constructed of ν. Therefore, G_{mod} allows only those c-planar embeddings that respect the connected clusters of C and takes the connected components of $G(\nu)$ into account. Hence the planar connectivity augmentation within G_{mod} has introduced a minimum cardinality augmenting edge set, connecting $G(\nu)$ such that the boundary of its external face in any planar drawing of $G(\nu)$ consists of a connected not simple cycle. Furthermore, the minimum cardinality augmenting edge set connects the connected components of the original $G(\nu)$, so that there exists at the most one edge between two connected components. Therefore $G_{mod} - G(\nu)$ is embedded in the outer face of $G(\nu)$ after planar connectivity augmentation. Hence, C is c-planar.

- "$\Rightarrow$" We have that C is c-planar. Thus C_{sub} is c-planar. Hence the vertices of $G(\nu)$ can be embedded so that there is a sequence of faces $f_1, \ldots, f_k$ with the following property: for all $1 \le i < k$, there is at least one vertex of W on the boundary between f_i and f_{i+1} and the boundaries of the faces f_i $(1 \le i \le k)$ contain all vertices of W.

Therefore G has a planar connectivity augmentation in respect to the vertices of $G(\nu)$.

□

Theorem 7.17. *Let $C = (G, T)$ be a connected clustered graph with a cluster tree T with only one level below the root cluster and ν its only non-root non-connected cluster. C can be tested for c-planarity in $O(n^2)$ time in respect to the number n of vertices of C.*

Proof. We can create the c-split clustered graph C_{sub} in $O(n)$ time where n is the number of vertices of G. The c-planarity testing can be done in $O(n^2)$ time and the planar connectivity augmentation of the subgraph induced by the non-connected clustered graph in $O(n)$ time. As a result the algorithm can be implemented in $O(n^2)$ time where n is the number of vertices of G. □

Algorithm 26: The algorithm for clustered graphs $C = (G, T)$ that contain a connected root cluster, a non-connected child cluster ν and an arbitrarily number of connected child clusters. It computes an embedding Π and the minimum cardinality augmenting edge set if C is c-planar.

```
Input: A clustered graph $C = (G, T)$ that contains a connected root cluster, a
       non-connected child cluster $\nu$ and an arbitrarily number of connected child
       clusters.
Result: true if and only if there is a $c$-planar connectivity augmentation for $\nu$; in
        the positive case an embedding $\Pi$ and the minimum cardinality augmenting
        edge set will be computed.
Compute $C_{sub}$ by splitting the non-connected cluster for each connected component
of the subgraph $G(\nu)$;
Apply the $c$-planarity test by Cohen, Eades and Feng [29, 30] on $C_{sub}$;
if the test return false then
    return false;
Apply on $G_{mod}$ (see Theorem 7.16) in respect to the vertices of $G(\nu)$ the planar
connectivity augmentation algorithm;
if a planar connectivity augmentation exists then
    Compute $\Pi$;
    return true;
else
    return false;
```

7.5 Multi-Level Clustered Graphs

We extend the algorithm of the previous section to clustered graphs with more than one level in the tree T. Consider a clustered graph $C = (G, T)$ with at least two non-connected clusters where G is connected. Then if for every non-connected cluster ν in the cluster tree T the parent cluster and all siblings of ν are connected, we show that it is possible to connect the non-connected clusters using the planar connectivity augmentation described in Chapter 6.

To do so, we compute the c-split clustered graph C_{sub} of C, described in Section 7.3. Then, we traverse T towards the root starting at the leaves in order to do the followings: For every non-connected cluster ν of C that has connected siblings μ and a connected parent $pa(\nu)$, we test whether the subgraph $G(pa(\nu))$ is planar, test whether the edges that are incident to $pa(\nu)$ can be drawn into the outside of the drawing of $G(pa(\nu))$ (see Figure 7.53) and test whether there exists a planar connectivity augmentation of the vertices of ν. If a

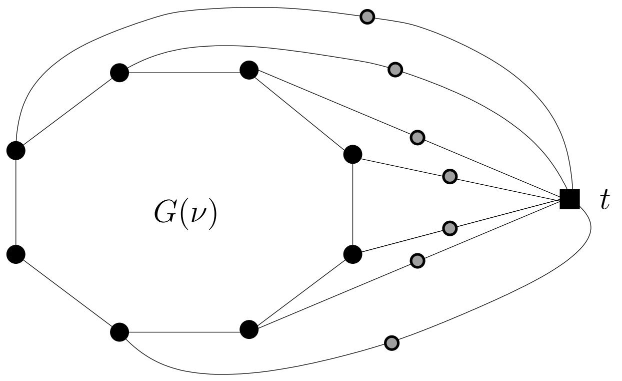


Figure 7.53: Constructing an auxiliary graph G_{mod} from the connected subgraph $G(\nu)$ where the incident edges of ν are connected with a dummy vertex t [30]

planar connectivity augmentation exists, then an embedding Π and a minimum cardinality augmenting edge set is computed and `true` is returned. Otherwise, C is not c-planar. If we get Π, we can apply the techniques used in the c-planarity embedding algorithm of Feng and Eades for the connected clusters using Π to obtain a c-planar embedding of C.

Theorem 7.18. *Let $C = (G, T)$ be a connected clustered graph where its non-connected clusters ν have a connected parent cluster and only connected sibling clusters. Let $C_{sub} = (G, T_{sub})$ be its c-split clustered graph. Let G_{mod} be the graph constructed by the c-planarity test of Feng, Eades and Cohen [29, 30] applied to $pa(\nu)$ of C_{sub} and where the subgraphs $G(chl(pa(\nu)))$ are replaced with their wheel graphs. If C_{sub} is c-planar and there exists for every non-connected cluster ν a planar connectivity augmentation for the vertices of the subgraph $G(\nu)$ in G_{mod} then C is c-planar.*

Proof. The proof is by construction. We extend the c-planarity test by Cohen, Eades and

Feng [29, 30] as follows: For every connected parent cluster $pa(\nu)$ with a non-connected child cluster ν, we construct the graph G_{mod} as stated in the theorem (see Fig. 7.53). Then we apply the linear time planarity test based on PQ-trees [8, 11] to G_{mod} and in the positive case the planar connectivity augmentation for the subgraph induced by the non-connected cluster (see Chapter 6).

As we do this recursively for every connected parent node that has a non-connected cluster by taking the wheel graphs as constructed in the c-planarity test into account, we can test c-planarity in $O(n^2)$ time where n is the number of vertices of G. □

Theorem 7.19. *Let $C = (G, T)$ be a connected clustered graph where its non-connected clusters ν have a connected parent cluster and only connected sibling clusters. C can be tested for c-planarity and in the positive case embedded in $O(n^2)$ time with respect to the number n of vertices of C.*

Algorithm 27: The algorithm for clustered graphs that contain a connected root cluster and non-connected clusters with a connected parent cluster and sibling clusters. It computes an embedding Π and the minimum cardinality augmenting edge set if C is c-planar.

```
Input: A clustered graph $C = (G, T)$ that contains a connected root cluster and
       non-connected clusters with connected parent cluster and sibling clusters.
Result: true if and only if there is a $c$-planar connectivity augmentation for $\nu$; in
        the positive case an embedding $\Pi$ and the minimum cardinality augmenting
        edge set will be computed.
Compute $C_{sub}$ by splitting the non-connected cluster for each connected component
of the subgraph $G(\nu)$;
Change the $c$-planarity test of Cohen, Eades and Feng [29, 30] as follows and apply
it to $C_{sub}$;
for every connected parent cluster of a non-connected cluster do
    Construct $G_{mod}$ as described in Theorem 7.18;
    Test planarity of $G_{mod}$;
    if the planarity test returns false then
        return false;
    Apply the subgraph induced planar connectivity augmentation algorithm for the
    vertices of $G(\nu)$ in $G_{mod}$ (see Chapter 6);
if a planar connectivity augmentation exists then
    Compute $\Pi$;
    return true;
else
    return false;
```

We note that this technique can be applied to connected clustered graphs $C = (G, T)$ where the non-connected clusters lie on the same path from the root to the leaves. This is done again by computing the c-split clustered graph C_{sub}, see Section 7.3 for details, for every connected component C_i of C, testing it for c-planarity. In this step, every connected component is modified with wheel graphs. Then we get graph G_{mod}.

We now traverse the path of non-connected clusters in the original cluster tree T from the leaves to the root and apply to the vertices of these clusters the planar connectivity augmentation algorithm (see Chapter 6) in G_{mod}. This can be done in $O(n^2)$ time where n is the number of vertices in the underlying graph G. In the positive case, an embedding can be computed in $O(n^2)$ time.

Finally, we consider two non-connected clusters that are siblings in an arbitrary clustered graph where all other clusters are connected and G is planar. If the two clusters are contained in two different connected components (if additionally the root is non-connected) or if they are contained in two different biconnected components or in two different subtrees of a BC-tree, we can apply the one-cluster-method for each connected or biconnected component independently. This can be extended to an arbitrarily number of non-connected clusters that are siblings under the condition that they are in different connected or biconnected components.

Observe that combining the techniques used in Sections 7.3 and 7.4, there is an even more larger class of clustered graphs that can be tested for c-planarity.

Chapter 8

Edge-Augmentation of c-Planar Clustered Graphs

Parts of this chapter have already been published in [54] and are joined work with Michael Jünger and Sebastian Leipert.

One could expect that once we have a c-planar embedded clustered graph $C = (G, T)$ at hand it may be augmented by general planar edge-augmention of the planar embedding of G.

For simplicity, we first consider edge-augmentation on the example of triangulation. A planar graph is triangulated (triangular or maximal planar) when every face has exactly three vertices. If a planar graph is not triangulated, then there exists a face f that has at least four different vertices, for example v_1, v_2, v_3 and v_4 in this order around the face. Hence G is planar, we can achieve a planar embedding. Since the edges (v_1, v_3) and (v_2, v_4) are not included in f, adding one of them and repeating this for all not triangulated faces, we receive a triangulated planar graph. It is assumed, that G is biconnected planar, otherwise an augmentation algorithm is used. The first triangulation algorithm is due to Read [70], and modified by De Freysseix such that it runs in linear space. It works as follows: For every pair of consecutive neighbors u and v of a current visited vertex v, we add an edge $e = (u, w)$ to G if w is not adjacent to u according to the planar embedding. Applying this to all vertices of G a triangulated graph is received that might contain multiple edges. The multiple edges can be replaced as follows: if we delete a multiple edge e we get a face with four vertices v_1, v_2, v_3, v_4. W.l.o.g. we assume that $e = (v_1, v_3)$, then we add an edge that connects v_2, v_4. Hagerup and Uhrig [44] modified that algorithm so that the resulting triangulated graph is simple. Kant and Bodlaender present in [7] a canonical triangulation as a good and simple method for computing a canonical ordering while triangulating the graph. He also give a proof of $\mathcal{NP}$-completeness for triangulating a planar graph while minimizing its maximum degree. Additionally, they introduce an approximation algorithm. Obviously, since a clustered graph is a generalization of a graph, all $\mathcal{NP}$-hard problems on graphs remain $\mathcal{NP}$-hard on clustered graphs and (hence $\mathcal{NP}$-hard on compound graphs).

The first algorithms that use triangulation for a c-planar clustered graph $C = (G, T)$ by triangulating planar G are presented in [25]. However, triangulation of a c-planar embedding by edge addition has not been investigated.

In Figures 8.1 and 8.2 two examples of triangulated c-planar embedded clustered graphs are given.

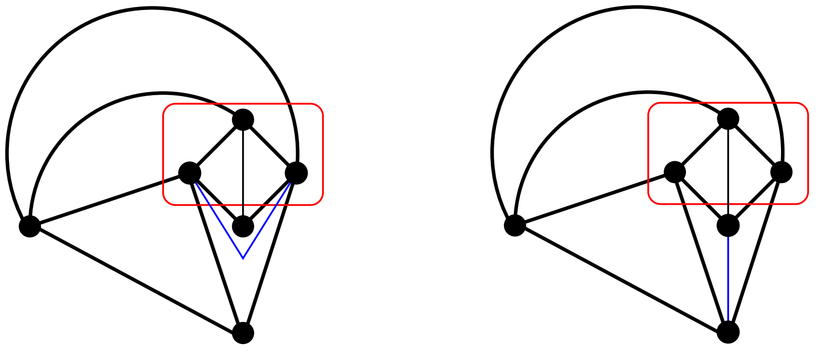

(a) Not feasible case (b) Feasible case

Figure 8.1: c-Planar embedded c-connected clustered graph $C = (G, T)$ visualized black with a red cluster; triangulation with edge colored blue destroys c-planarity in a) but preserves it in b)

In Figure 8.1, we have a c-connected clustered graph $C = (G, T)$ with a c-planar embedding visualized black. Additionally, C is triangulated with an edge visualized blue. Observe, that the blue edge e has end vertices in the cluster colored red and therefore should belong to that cluster.

Unfortunately, in Figure 8.1(a) e encapsulates a vertex that belongs to the root cluster. Additionally, there is no way to embed e into an other face since G is triconnected. Hence, C is not c-planar after the addition of e and therefore adding e is not feasible. On the other hand, in Figure 8.1(b) shows a feasible edge addition that preserves the c-planar embedding of C.

In Figure 8.2, we have a clustered graph $C = (G, T)$ without a cut cluster and with a c-planar embedding also visualized black. While the edge addition in Figure 8.2(a) produces a non-repairable edge-region crossing the edge addition in Figure 8.2(b) keeps the c-planar embedding.

Consequently, we cannot extend a c-planar clustered graph by simply extending G. Naturally, the question arises whether there exists a c-planar clustered graph class that is resistant in concerning to general planar edge-augmentation. Fortunately, we may answer this request positively. We assume from now on that every non-trivial cluster of C contains at least two vertices of G.

Theorem 8.1. *Let $C = (G, T)$ be a c-connected c-planar clustered graph without any cut*

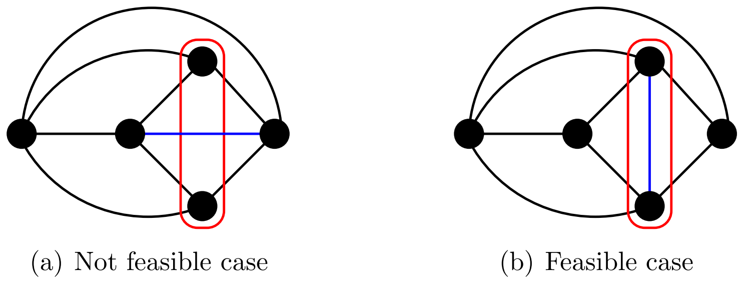


Figure 8.2: c-Planar embedded clustered graph $C = (G, T)$ without a cut cluster visualized black with a red cluster; triangulation with edge colored blue destroys c-planarity in a) but preserves it in b)

cluster. C remains c-planar after edge-augmentation if and only if G remains planar after edge-augmentation.

Proof. A c-connected clustered graph is c-planar if G is planar and for all clusters ν, $G - G(\nu)$ can be drawn outside of the drawing of $G(\nu)$ (see Theorem 2.3). If additionally $G - G(\nu)$ is connected for all clusters ν, it is obvious that $G - G(\nu)$ can be drawn outside of $G(\nu)$ for all clusters ν. As we add edges to C, we do not destroy the properties of C. Therefore, for all clusters ν, $G(\nu)$ is still connected, and $G - G(\nu)$ is also still connected. Therefore, C is still c-planar. As mentioned in [41, 40], we destroy the c-planar embedding if we introduce edges to a connected cluster μ of T that isolates vertices belonging to a cluster ν not equal to μ. But this only happens, if one of the following cases is true:

1. Either ν is an ancestor of μ, but then $G - G(\mu)$ is not connected before augmentation, or

2. ν is no ancestor of μ, but then $G(\nu)$ is not connected before augmentation.

If $G(\nu)$ is connected, we destroy the c-planar embedding, if we isolate $G(\nu)$ by adding edges to a connected cluster μ not descendant of ν around the drawing of $G(\nu)$. But this only happens if a connected component of $G - \{G(\nu) \cup G(\mu)\}$ is isolated to $G(\nu)$. But this means that $G - G(\mu)$ is not connected in the c-planar embedding before adding the edges what leads to a contradiction. □

Consequently, whenever we can augment a given planar clustered graph $C = (G, T)$ to a c-connected planar clustered graph without any cut cluster, C is c-planar and keeps its c-planarity by planar edge-augmentation.

Let $\mathcal{C}$ be the class of c-planar c-connected clustered graphs without any cut cluster.

Naturally, the question arises how hard it is to augment a planar clustered graph C such that the final C belongs to $\mathcal{C}$. Unfortunately, the complexity status of this problem is still

```
Algorithm 28: Augmentation of a c-planar embedded c-connected clustered graph $C$.
    Input: c-connected clustered graph $C = (G, T)$ with a c-planar embedding $\Gamma$.
    Result: edge-augmented $C$ that is c-planar and has the properties of the used planar
            edge-augmentation.
    Augment $\Gamma$ by edge-addition s. t. resulting $C' = (G', T) \in \mathcal{C}$;
    Augment planar embedding of $G'$ by general approach;
```

unsolved. Clearly, a c-planar clustered graph can always be edge-augmented such that the resulting c-planar clustered graph belongs to $\mathcal{C}$.

In this chapter, we focus on clustered graphs with a given c-planar embedding. How to augment a c-planar embedded clustered graph with a given c-planar embedding in polynomial time such that it is c-connected is presented by Feng [29]. Hence we may assume that we have a c-connected clustered graph with a c-planar embedding at hand. In the following we consider the complexity of edge-augmentation with the minimum number of edges such that the final C belongs to $\mathcal{C}$.

Kant and Bodlaender [7] studied augmentation problems for planar graphs. Especially, they showed that the minimum planar biconnectivity edge-augmentation problem is $\mathcal{NP}$-complete.

Theorem 8.2 (Kant,Bodlaender [7]). *The problem of deciding whether adding at most K edges to a connected planar graph $G = (V, E)$ can lead to a biconnected planar graph is $\mathcal{NP}$-complete.*

Recall that a clustered graph is cluster-biconnected if it has no cut cluster and no cut vertex.

Since biconnectivity is a specification of cluster-biconnectivity:

Corollary 8.1. *The problem of deciding whether adding at most K edges to a c-planar c-connected clustered graph $C = (G, T)$ can lead to a cluster-biconnected c-planar clustered graph is $\mathcal{NP}$-complete.*

Proof. Follows immediately by Theorem 8.2 since a graph can be seen as a clustered graph with only one cluster, the root-cluster. □

Observe that this is even true when restricted to cut clusters. We transform each planar graph instance I_G to a clustered graph instance I_C by replacing each cut vertex by a star, and assigning the star to a cluster. Then I_G can be edge-augmented by K edges to a biconnected graph if and only if I_C can be edge-augmented by K edges to a clustered graph without any cut clusters.

Corollary 8.2. *The problem of deciding whether adding at most K edges to a c-planar c-connected clustered graph $C = (G, T)$ such that for every cut cluster ν $G - G(\nu)$ is connected is $\mathcal{NP}$-complete.*

Fortunately, it turns out that we do not need the minimum number of edges for our edge-augmentation problem: for every cut cluster ν it is sufficient to add edges to $G - G(\nu)$ such that the edge addition forms a tree. This follows directly from the characterization made in Chapter 5. Consequently, it is sufficient to consider a heuristic algorithm to augment a given c-planar embedded c-connected clustered graph to a clustered graph of $\mathcal{C}$. Next we develop such an algorithm that has linear running time.

First we consider the problem of deciding whether a given clustered graph belongs to $\mathcal{C}$.

Recall that having two vertices v, w of G at hand, we define the lowest common ancestor (lca) of v, w to be the ancestor of v, w in T that is the farest away from the root cluster in T. Recall that the greatest uncommon ancestors (gua) of v, w we define to be the children of $\texttt{lca}(v, w)$ on the two (unique) paths from v and w to $\texttt{lca}(v, w)$ in T. Recall that if v, w are connected by an edge e, we define $\texttt{lca}(e) := \texttt{lca}(v, w)$.

The level graphs are constructed as follows (a similar construction is made by Feng [29]): For each cluster ν of T we define a shrink graph in respect to ν

$$G_{\texttt{shrink}}|_\nu = (V_{\texttt{shrink}}|_\nu, E_{\texttt{shrink}}|_\nu)$$

with the vertex set

$$V_{\texttt{shrink}}|_\nu = \{\mu \in T \mid \mu \text{ is a child of } \nu \text{ in } T\} \cup \{d\}$$

with a dummy vertex d and the edge set

$$E_{\texttt{shrink}}|_\nu = \{e \in E \mid \texttt{lca}(e) = \nu\} \cup \{(v, d) \mid (v, w) \text{ are incident edges of } \nu \text{ in } T\} \cup \{(d, d)\}.$$

Observe that the vertices of $V_{\texttt{shrink}}|_\nu$ correspond to (trivial or non-trivial) child clusters of ν in T.

Further, the cut vertices in $G_{\texttt{shrink}}$ correspond to cut clusters and cut vertices in C. Hence, C is cluster-biconnected if and only if all shrink graphs has no cut vertex (beside d) that corresponds to a cut cluster. Consequently, either we augment each shrink graph to biconnectivity or we adapt the biconnectivity augmentation such that it consults only the cut vertices that correspond to cut clusters in C.

Consequently, a clustered graph C is cluster-biconnected if and only if for every cluster ν its shrink graph has no cut vertices beside d. Therefore, cluster-biconnectivity can be tested in linear time by testing biconnectivity of every shrink graph of C. Observe that we can test the weaker version of cluster-biconnectivity, the co-connectivity, easily that restricts on cut clusters.

Next we consider how to augment c-planar embedded c-connected clustered graphs in order to make them additionally cluster-biconnected.

We assume that the given clustered graph $C = (G, T)$ is c-connected and c-planar embedded.

Algorithm 29: Augmentation of a c-planar embedded c-connected clustered graph C to a clustered graph of $\mathcal{C}$.

```
Input: $c$-connected clustered graph $C = (G, T)$ with a $c$-planar embedding $\Gamma$.
Result: edge-augmented $\Gamma$ s. t. resulting $C' \in \mathcal{C}$.
foreach cluster $\nu$ top down level by level in $T$ do
    Calculate $G_{\text{shrink}}|_\nu$ according to $\Gamma$;
    Apply planar biconnectivity edge-augmentation on the resulting planar embedding $\Gamma|_\nu$ of $G_{\text{shrink}}|_\nu$;
    Introduce added edges of $G_{\text{shrink}}|_\nu$ to $\Gamma$ according to $\Gamma'$;
```

We seek to augment the given c-planar embedding of C. We compute the shrink graph for each cluster ν of C and apply the planar biconnectivity edge-augmentation on them (see Algorithm 29).

In Figure 8.3 we show an example how this edge-augmentation works. While C does not have any cut cluster (and therefore no edge-addition is required for the weaker version of cluster-biconnectivity, the co-connectivity, that consider only cut clusters), edges e_1 and e_2 are added for achieving cluster-biconnectivity. Observe that the minimum number of edges to be added in this case would be 1 since it is sufficient to add e_2.

By Theorem 8.1 the resulting c-planar embedding remains c-planar if and only if it is planar. Consequently, we can apply any planar augmentation graph approach on the resulting cluster-biconnected embedding, since it implies co-connectivity.

Theorem 8.3. *Let $C = (G, T)$ be a c-planar embedded c-connected clustered graph. C can be tested on cluster-biconnectivity in linear time concerning the number of vertices of G. If C is not cluster-biconnected, it can be augmented to a cluster-biconnected c-planar embedded clustered graph in linear time.*

Proof. Since the original edges appear in at most constant number of shrinked graphs and the biconnectivity augmentation for an embedded graph can be done in linear time, we have a linear running time in total. □

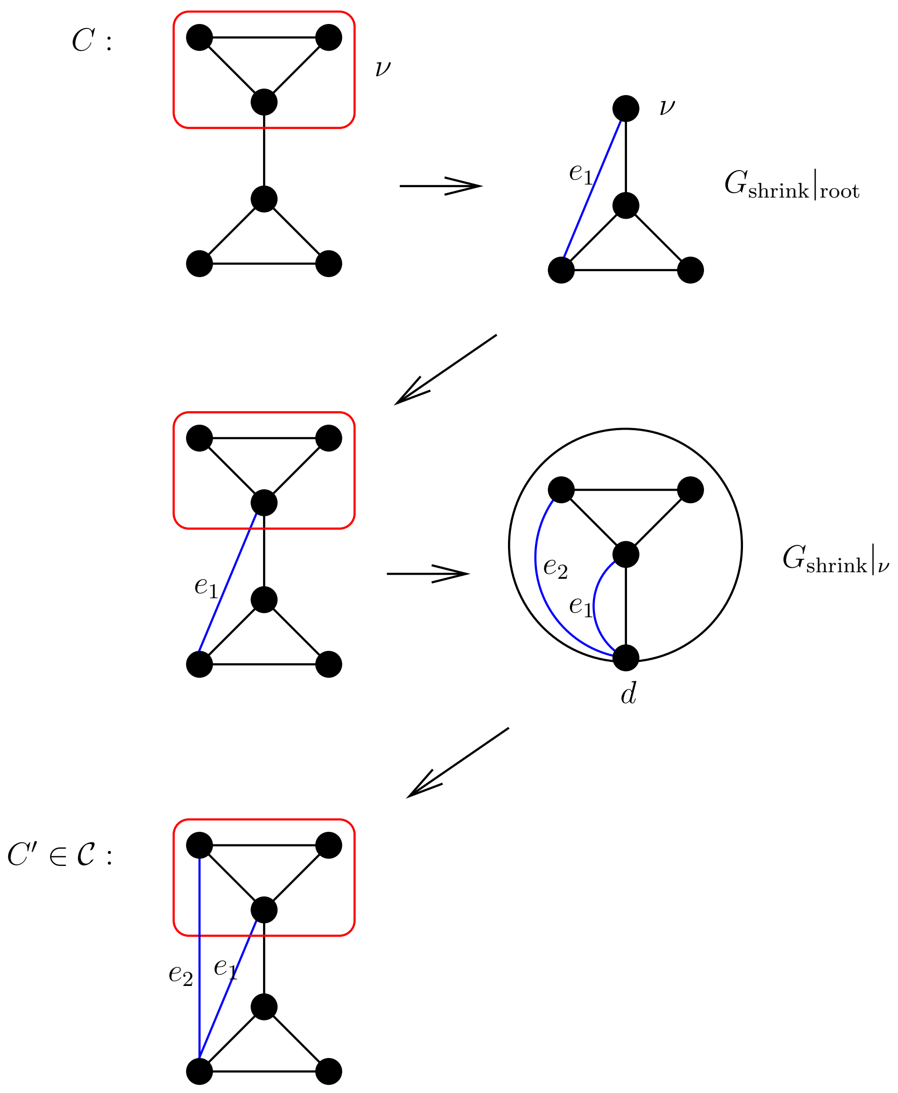


Figure 8.3: Example of calculating cluster-biconnectivity of a c-planar clustered graph C with resulting clustered graph C'; on the left-hand side the modification in the embedding of C is visualized while on the right-hand side the shrink graphs are presented - upper shrink graph corresponds to the root cluster while the lower represents the cluster ν, edges colored blue are added edges

Part II

Minimum Edge Deletion and Crossing Minimization

Chapter 9

Minimum Edge Deletion

The results of this chapter are based on joint work with Elisabeth Gassner and are available as technical reports (see [36] and [37]).

In Section 9.1 we show that the maximum planar subgraph problem remains $\mathcal{NP}$-complete even for graphs without a $K_{3,3}$ or K_5 minor, respectively.

In Section 9.2 we investigate the problem of finding a minimal weighted set of edges whose removal results in a graph without minors that are contractible onto a prespecified set of vertices. Such minors are called rooted. The problem of a minimal weighted deletion of all rooted $K_{1,3}$-minors and $K_{2,3}$-minors, respectively, is proved to be $\mathcal{NP}$-hard for general graphs. Furthermore, an $\mathcal{O}(n^3)$ time algorithm is developed for the rooted $K_{1,3}$-minors deletion problem on planar graphs while for the rooted $K_{2,3}$-minor planar graph a characterization is presented.

9.1 Maximum Planar Subgraph ($\mathcal{NP}$-hardness proofs)

The results of this section are joint work with Elisabeth Gassner.

Wagner characterizes a planar graph as a graph that has no K_5 and $K_{3,3}$ minors [84]. His theorem is a significant reformulation of Kuratowski's well-known result [58]. If G has either no K_5 or no $K_{3,3}$ minor it is intuitively close to planarity.

The *maximum planar subgraph problem* (MPSP for short) is well-studied: Given a graph $G = (V, E)$ and a positive integer $k \leq |E|$, is there a subset $E' \subseteq E$ with $|E'| \geq k$ such that the graph $G' = (V, E')$ is planar?

Liu et al., Yannakatis, and Watanabe et al. all independently showed that this problem is $\mathcal{NP}$-complete [64, 85, 86].

The weighted version of MPSP is a generalization of MPSP in which weights are assigned to the edges and the task is to find a planar subgraph of maximum total weight. Recently, Faria, de Figueiredo, and de Mendonça have shown that the maximum planar subgraph problem remains $\mathcal{NP}$-complete for cubic graphs [28]. For a survey on the maximum planar subgraph problem, the reader is referred to [63].

Obviously, the class of non-planar cubic graphs is not equal to the class of graphs that are either K_5-free or $K_{3,3}$-free (see Figure 9.1).

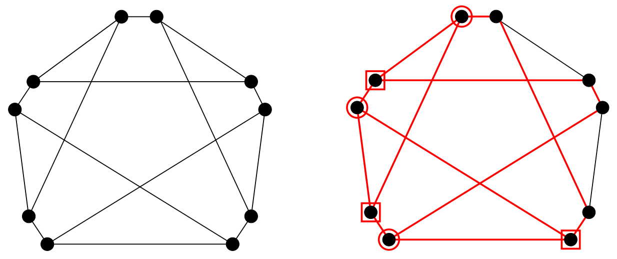

(a) Cubic graph (b) $K_{3,3}$ minor

Figure 9.1: A non-planar cubic graph that has minors isomorphic to K_5 and to $K_{3,3}$

Throughout this section Tutte connectivity is used.

In this section we prove $\mathcal{NP}$-completeness of the maximum planar subgraph problem on K_5-free or $K_{3,3}$-free graphs, respectively.

Clearly the problem is in $\mathcal{NP}$ and we may obviously reduce the problem on connected graphs.

We use the following lemma for our $\mathcal{NP}$-hardness proofs.

Lemma 9.1 (Truemper [82]). *If G is a 2-sum of G_1 and G_2, then for any 3-connected minor N of G, G_1 or G_2 has a minor isomorphic to N.*

Proof. If this is not so, then N has a 2-separation induced by the 2-separation that is given by G_1 or G_2 minus their connecting edges. This leads to a contradiction to the fact that N is 3-connected. $\square$

Let $\mathcal{K}_5$ be the following class of graphs, each constructed as follows. Let $G = (V, E)$ be a connected planar graph and E' a nonempty subset of E.

We apply an iterative processing of the edges e of E': We take the 2-sum of the current graph G with K_5, i.e. $G \oplus_2 K_5$, where $e \in E'$ is the edge of G involved in the 2-sum. Then, we redefine G to be the 2-sum.

We name $G[K_5]$ to be the final 2-sum that results from this iterative process.

Further, we define $\mathcal{K}_{3,3}$ and $G[K_{3,3}]$ analogiously way, using $K_{3,3}$ instead of K_5.

Theorem 9.1. *The maximum planar subgraph problem is $\mathcal{NP}$-complete for the two classes $\mathcal{K}_5$ and $\mathcal{K}_{3,3}$.*

Given a connected planar graph $G = (V, E)$ the connected vertex cover decision problem (CVC for short) asks for a vertex cover N in G of cardinality at most k, such that the subgraph induced by N is connected. CVC is known to be $\mathcal{NP}$-hard [33].

The proof of Theorem 9.1 uses the following result by Asano:

Theorem 9.2 (Asano [1])**.** *Let $G = (V, E)$ be a connected planar graph, and $G(2)$ obtained of G by splitting each edge of G once. The new vertices are denoted by a_i with $i = 1, 2, 3, \dots, |E|$. Let G_2 be the graph constructed by a planar embedding of $G(2)$: for every face f add edges between two a_i, a_j, $i \neq j$, if they belong to the boundary of f and are adjacent to the same vertex. Let G_2^* be the dual graph of G_2 with the modified planar embedding of $G(2)$. Let $N \subset V$ be a connected vertex cover of G with $|N| \leq k$, $k \in \mathbb{N}$.*

If G has a connected vertex cover of size at most k, then $G(2)$ has a Steiner tree T for the terminal set $A = \{a_i \mid i = 1, 2, \dots, |E|\}$ where $|E(T)| \leq k$ and all edges $(a_i, a_j)^* \in F^*$ with $i, j = 1, 2, 3, \dots, |E|$, $i \neq j$ in $G_2^* - E(T)^*$ are coloops.

On the other hand, let $S \subset E(G_2^*) - F^*$ be a subset of edges in G_2^* with $|S| \leq k$ such that all edges $(a_i, a_j)^* \in F^*$ with $i, j = 1, 2, 3, \dots, |E|$, $i \neq j$ in $G_2^* - S$ are coloops, then G has a connected vertex cover of size at most k.

Asano's result [1] implies that the following multi-cut problem (MC for short) is $\mathcal{NP}$-complete.

Corollary 9.1. *Given a connected planar graph $H = (V, E)$ with edge partition $E = E_1 \uplus E_2$ and an integer k, deciding whether there exist a subset $S \subset E_1$ with $|S| \leq k$ such that all edges $e \in E_2$ are coloops in $H - S$ is $\mathcal{NP}$-complete.*

Observe that MC is related to the minimum multi-cut problem [19] (MMC for short): Given is a graph $G = (V, E)$, a set $S \subseteq V \times V$ of source-terminal pairs, $k \in \mathbb{N}$ and a weight function $w : E \to N$. Is there a multi-cut, i.e., a set $E' \subseteq E$ such that the removal of E' from E disconnects s_i from t_i for every pair $(s_i, t_i) \in S$ such that $\sum_{e \in E'} w(e) \leq k$?

Therefore, MC seeks for a minimum multi-cut in E_1 whether MMC uses E. Hence MC is equal to MMC if and only if $E_2 = \emptyset$. Note that MMC is a generalization of the minimum multiway cut and is $\mathcal{NP}$-hard even when the graph is a tree [19, 35]. For a survey and bibliography the reader is referred to [4, 17].

Proof of Theorem 9.1. We show that there exists a polynomial reduction of MC (that is $\mathcal{NP}$-complete by Corollary 9.1) to the maximum planar subgraph problem on graphs of the classes $\mathcal{K}_5$ or $\mathcal{K}_{3,3}$, respectively.

Given an instance of MC, i.e., a planar graph $G = (V, E_1 \uplus E_2)$ and an integer k, we can construct an instance of the weighted MSPS for graphs of $\mathcal{K}_5$ or $\mathcal{K}_{3,3}$, respectively: We set

$E' = E_2$ and create iteratively an instance $G[N]$ of $\mathcal{K}_5$ or $\mathcal{K}_{3,3}$, respectively, with $N = K_5$ or $N = K_{3,3}$, respectively.

Moreover, we define a weight function for the edges of $G[N]$: For each edge e of $G[N]$ that is also included in G, i.e., $e \in E_1$, we set $c(e) = 1$; otherwise $c(e) = k + 1$.

Claim: Let $G = (V, E_1 \uplus E_2)$ be a connected, planar graph, $N \in \{K_{3,3}, K_5\}$, $E' = E_2$ and let $S \subseteq E_1$. Then $G[N] - S$ does not contain any N-minor if and only if all edges $e \in E_2$ are coloops in $G - S$.

Proof of claim: First assume that there exists an edge $e \in E_2$ such that $e = (i, j)$ is not a coloop in $G - S$. Then there exists a path P from i to j in $G - S$ that does not contain edge e. Since $E_2 = E'$ and hence e is involved into a 2-sum with $G - S$ and N, $G[N] - S$ contains an N-minor using path P instead of e. This leads to a contradiction to the assumption that $G[N] - S$ is N-free.

Now assume that there exists an N-minor in $G[N] - S$. Since $G - S$ is planar and hence N-free we conclude that there is an edge $e = (i, j) \in E' = E_2$ that is involved into a 2-sum of $G - S$ and N. Furthermore, since $G[N] - S$ is not planar there is a path P in $G - S$ from i to j. This contradicts the assumption that e is a coloop in $G - S$. This proves the claim.

Our claim implies that there exists a feasible solution $S \subset E_1$ with $|S| \leq k$ of instance $G = (V, E_1 \uplus E_2)$ for MC if and only if there exists a subset of edges S' of $G[N]$ with total weight $c(S') \leq k$ whose removal yields a planar subgraph. Observe that $c(e) = k+1 > c(S')$ for $e \in E' = E_2$ and hence $S' \subseteq E_1$.

Moreover, if we replace every edge e in $G[N]$ with $c(e) = k + 1$ by $(k + 1)$ copies of edge e we conclude the theorem. □

Finally, we get the following result.

Corollary 9.2. *The maximum planar subgraph problem is $\mathcal{NP}$-complete for the following classes of graphs.*

1. *The graphs without a K_5 minor,*

2. *The graphs without a $K_{3,3}$ minor.*

Proof. Clearly the problem is in $\mathcal{NP}$ since planarity is polynomially checkable.

The $\mathcal{NP}$-completeness follows immediately by Lemma 9.1: The class of graphs without a K_5 or $K_{3,3}$ minor, respectively, contains the class $\mathcal{K}_{3,3}$ or $\mathcal{K}_5$, respectively, for which Theorem 9.1 establishes the problem to be $\mathcal{NP}$-complete. □

Surprisingly, the (weighted) maximum planar subgraph problem is easy for triconnected non-planar graphs G without a minor isomorphic to $K_{3,3}$. By the major decomposition theorems in [81], G is then isomorphic to K_5. Hence, the maximum planar subgraph of G is equal to K_5 minus one of the cheapest edges.

We consider a triconnected non-planar graph G without a minor isomorphic to K_5. G is called a Δ-*sum* (composition) of G_1 and G_2, denoted $G = G_1 \oplus_\Delta G_2$, if identification of an arbitrary triangle of G_1 with an arbitrary triangle in G_2 and subsequent deletion of the edges of this triangle produces G. By the major decomposition theorems in [81], G is then either isomorphic to $K_{3,3}$, or to V_8, or is equal to a Δ-sum composition of planar graphs. For the first two cases, the (weighted) maximum planar subgraph problem is easy: we delete one of the cheapest edges in $K_{3,3}$ or V_8, respectively. For the remaining case we conjecture that it is $\mathcal{NP}$-complete as well.

Furthermore, we conjecture that the crossing minimization problem on graphs without a K_5 or $K_{3,3}$ minor, respectively, is $\mathcal{NP}$-hard.

9.2 Rooted Minor Deletion

The results of this section are joint work with Elisabeth Gassner. While the results of Section 9.2.2 are based mainly on Elisabeth Gassner's ideas, the results of Section 9.2.3 are mainly due to the author.

Many combinatorial optimization problems can be stated as minor deletion problem. Consider for instance the following problem: Given a graph $G = (V, E)$, find a subset of edges of minimal cardinality whose deletion results in a graph without C_1-minors (where C_1 is a loop). This minor-deletion problem is equivalent to the maximum spanning tree problem which can be solved in polynomial time. On the other hand there are several minor-deletion problems that are $\mathcal{NP}$-hard such as the maximum planar subgraph problem or the longest path problem.

The characterization of graphs with special properties by means of minor exclusion has gained increasing attention in recent years. A famous theorem in this area is Wagner's reformulation of Kuratowsky's theorem: A graph is planar if and only if it has no minor isomorphic to K_5 or $K_{3,3}$ [84]. A similar characterization holds for outerplanar graphs: A graph is outerplanar if and only if it has no minor isomorphic to K_4 or $K_{2,3}$. If a graph class can be fully described by excluding a set of forbidden minors two natural problems occur: Let a graph G and a set of forbidden minors $\mathcal{F}$ be given:

1. Decide whether G contains a forbidden minor of $\mathcal{F}$.
2. Delete a minimum number of edges whose deletion results in a graph without any forbidden minor of $\mathcal{F}$.

The first problem was considered by Robertson and Seymour [73]. They proved that, for any class of graphs closed under isomorphism and taking minors, there exists an $O(n^3)$ time algorithm to decide if an input graph on n vertices belongs to the class (but generally we do not *know* the algorithm). Hence, this problem can be solved in strongly polynomial time. However, the second problem is not in $\mathcal{P}$in general. Watanabe, Ae and Nakamura [85] proved $\mathcal{NP}$-hardness for the minor-deletion problem on general graphs if $\mathcal{F}$ is a nonempty class consisting of triconnected graphs and there are arbitrary large graphs without minors of $\mathcal{F}$. Asano [1] strenghted the above result and showed that the edge-deletion problem is $\mathcal{NP}$-hard even on planar graphs if $\mathcal{F}$ describes a property π which is nontrivial and determined by the weighted triconnected components of a graph. This class of edge-deletion problems includes properties π such as series-parallel, outerplanar, co-outerplanar, ladder, or without cocycles of cardinality at least three.

In this chapter we will be concerned with the deletion of complete minors and complete bipartite minors. As we have already mentioned above, the maximum planar subgraph problem is to find a subset of edges of minimum cardinality whose deletion results in a graph without $K_{3,3}$-minors and K_5-minors. Liu and Geldmacher [64] proved the maximum planar subgraph problem to be $\mathcal{NP}$-hard. As we have seen in Section 9.1, this remains true even when restricted to K_5- or $K_{3,3}$-minor free, respectively, non-planar graphs.

If $K_{3,3}$ is replaced by $K_{2,3}$ and K_5 by K_4 we get a characterization of outerplanar graphs. Yannakakis [87] proved the maximum outerplanar subgraph problem to be $\mathcal{NP}$-hard on general graphs and Asano [1] proved $\mathcal{NP}$-hardness even for planar graphs.

Finally, the problem of deleting all $K_{1,3}$-minors and K_3-minors is equivalent to the problem of finding a maximum subgraph with maximum degree two and without cycles. By reduction from the Hamiltonian path problem it can be shown that this problem is also $\mathcal{NP}$-hard.

In order to understand the reason of the $\mathcal{NP}$-hardness of these problems we turn our attention to the problem of only excluding the corresponding bipartite minor.

The problem of deleting all $K_{1,3}$-minors is given by a (planar) graph $G = (V, E)$. The task is to find a maximum subgraph $G' = (V, E \setminus E')$ that does not contain any $K_{1,3}$-minor. Observe that a graph G' contains a $K_{1,3}$-minor if and only if G' contains a vertex $v \in V$ with $\deg(v) \geq 3$. Hence, the $K_{1,3}$-minor deletion problem boils down to finding a maximal subgraph whose maximum degree is equal to 2, i. e., a simple 2-matching. This problem can be solved in $O(\sqrt{n}m)$ time (in the unweighted case) and in $O(mn\log(n))$ or $O(n^3)$ time (in the weighted case) by using the algorithms of Gabow [32]. On the other hand, excluding all K_3-minors is equivalent to the problem of finding a maximal subgraph without minors that are cycles of length greater or equal to three which leads to the maximal spanning tree problem (if the graph is simple). Hence, the $K_{1,3}$-minor deletion problem as well as the K_3-minor deletion problem are solvable in polynomial time while the deletion problem of both minors becomes $\mathcal{NP}$-hard.

However, the behavior of the deletion problems corresponding to the minors $K_{2,3}$ and K_4 is quite different. Using the notation of Asano [1], the property of a graph to be $K_{2,3}$-minor free is determined by the weighted triconnected components of the graph. The same holds for the property to be K_4-minor free (i. e., series-parallel). Hence, both the $K_{2,3}$-minor deletion problem as well as the K_4-minor deletion problem are $\mathcal{NP}$-hard even for

2-connected planar graphs [1].

Our contribution to this topic deals with a much weaker minor definition, the so-called rooted bipartite minors, i. e., minors that can be contracted onto prespecified vertices. Rooted minors have already been investigated in order to prove the existence of minors. For instance, Robertson, Seymour and Thomas [72] proved the following observation: Given a graph G and four vertices v_1, v_2, v_3 and v_4, then either there exists a K_4-minor which is contractible onto $v_1, \ldots, v_4$ or G is planar with the four vertices on the same boundary. This observation was crucial for their proof of Hadwiger's Conjecture for K_6-free graphs. And Jørgensen [52] investigated rooted minors in order to prove that a 4-connected $K_{4,4}$-minor free graph on n vertices has at most $4n-8$ edges, which results in the observation that $K_{4,4}$-free graphs have vertex-arboricity of at most 4. Recently, Jørgensen and Kawarabayashi [53] proved extremal results on the minimum number of edges which guarantees the existence of a rooted $K_{3,4}$-minor (and $K_{3,3}$- and $K_{2,3}$-minor, respectively).

Our motivation to this topic is to study $\mathcal{NP}$-hard deletion problems, e.g., the maximum planar subgraph problem, under certain decompositions. The question is how to boil down, e.g. the maximum planar subgraph problem of a given graph, to its decomposition primes. Beside the common minimum cut problem that comes up while trying to reduce the connectivity in a (Tutte) 2-separation, we observe that in the case of (Tutte) 3-separation there occurs an additional connectivity reduction problem [57] that is equal to the problem of minimum deletion of rooted $K_{i,3}$-minors with $i \in \mathbb{N}$. Consequently, the results of this chapter are first steps towards extracting new polynomial sub-cases and gaining new insight in order to improve heuristics and exact methods.

In this chapter (see [37] for a technical report) we apply the edge-deletion approach to rooted $K_{i,3}$-minors with $i = 1, 2$. In Section 9.2.1 we introduce some definitions and notations. Section 9.2.2 is dedicated to the problem of deleting rooted $K_{1,3}$-minors. In contrast to the $K_{1,3}$-minor deletion the problem of finding a subset of edges of minimum weight whose deletion results in a graph without any rooted $K_{1,3}$-minor is proved to be $\mathcal{NP}$-hard on general graphs. However, for the special case of planar graphs we suggest an efficient algorithm which solves the problem in $O(n^3)$ time (where n is the number of vertices of the planar graph). Finally, in Section 9.2.3 the problem of minimum deletion of all rooted $K_{i,3}$-minors for a fixed $i \in \mathbb{N}$ is proved to be $\mathcal{NP}$-hard on general graphs. This explains why it is so difficult to achieve e.g. a maximum planar subgraph in general even under decomposition. For planar graphs that have no rooted $K_{2,3}$-minor we give a detailed characterization, while the complexity of the deletion problem remains open.

9.2.1 Problem Statement

Throughout this section we will use the following additional notation:

Let G be a planar graph and let Z be a face in G. Then the set of vertices of the boundary of Z is denoted by $\mathrm{bd}(Z)$.

A *k-star* is a tree in which all vertices except one are leaves and one vertex has degree k.

The vertex with degree k is called the *root* of the k-star.

Let G be planar and let Z be a face in G with $|\mathrm{bd}(Z)| = k$. The graph that results by adding a k-star to G such that the leaves of the k-star coincide with the vertices on the boundary of Z is denoted by $G \circledast Z$. Observe that $G \circledast Z$ is still planar. The root of the inserted k-star is denoted by $r(Z)$.

The contraction of a set of edges $L \subseteq E$ in a graph G is denoted by G/L. A graph resulting from adding an edge e to graph G is denoted by $G + e$.

Given a path P that contains the vertices i and j we denote the subpath of P from i to j by $P(i, j)$.

Finally, given an edge weight function $\ell : E \to \mathbb{R}$ and a subset of edges $L \subseteq E$ the total weight of L is defined by $\ell(L) = \sum_{e \in L} \ell(e)$.

Let $X = \{x_1, \ldots, x_p\}$ be a set of p vertices of G. Then G contains a rooted K_p-minor with respect to X ($K_p[X]$-minor for short) if K_p is a minor of G and K_p is contractible onto X, i. e., there exists a subgraph H' of G and a partition $V(H') = V_1 \uplus \cdots \uplus V_p$ of its vertex set into connected subsets such that contracting each of $V_1, \ldots, V_k$ yields a graph isomorphic to K_p and $x_i \in V_i$ for $i = 1, \ldots, p$.

Bipartite rooted minors are defined in a similar way. Let $X = \{x_1, \ldots, x_q\}$ be a set of vertices of G. Then G contains a rooted $K_{p,q}$-minor with respect to X ($K_{p,q}[X]$-minor for short) if there exists a bipartite subgraph H' of G and a partition $V(H') = U_1 \uplus U_p \uplus W_1 \uplus W_q$ of its vertex set into connected subsets such that contracting each $U_1, \ldots, U_p, W_1, \ldots, W_q$ yields a graph isomorphic to $K_{p,q}$ and $x_i \in W_i$ for $i = 1, \ldots, q$.

An instance of the minimum weighted rooted $K_{1,3}$-minor deletion problem (K13-DEL for short) is given by an edge weighted graph $G = (V, E, \ell)$ and three prespecified vertices $a, b, c \in V$. These three vertices are called terminals. Using the notation from above we are interested in deleting all $K_{1,3}[X]$-minors for $X = \{a, b, c\}$. Hence, the task is to find a subset of edges $E' \subset E$ with minimum total weight such that $G' = (V, E \setminus E')$ does not have any $K_{1,3}[X]$-minor with $X = \{a, b, c\}$.

The minimum weighted rooted $K_{2,3}$-minor deletion problem (K23-DEL for short) is defined analoguesly.

9.2.2 The Rooted $K_{1,3}$-minor Deletion Problem

In this section we investigate the problem of deleting edges with minimum total weight such that the remaining graph does not contain any rooted $K_{1,3}$-minor. We will first prove $\mathcal{NP}$-hardness for general graphs and then develop an efficient polynomial time algorithm for the problem on planar graphs.

K13-DEL can be reformulated in the following way: Given a graph $G = (V, E)$ and three terminals a, b, c. Find a subset of edges E' at minimum total weight whose deletion results in a graph $G' = (V, E \setminus E')$ with the following property: In G' there does not exist any vertex $v \in V \setminus \{a, b, c\}$ such that v can reach a, b and c along paths that do not contain the terminals a, b or c as intermediate nodes.

K13-DEL on General Graphs

In this section we show that K13-DEL is strongly $\mathcal{NP}$-hard on general graphs.

Theorem 9.3. *Given a graph $G = (V, E)$ and a set $X \subset V$ with $|X| = 3$. The problem of finding a subset of edges with minimum weight such that the resulting graph does not contain any $K_{1,3}[X]$-minor is $\mathcal{NP}$-hard.*

The proof is done by a reduction from Max-Cut. Let $\bar{G} = (\bar{V}, \bar{E})$ be an instance of Max-Cut with $|\bar{V}| = n$ and $|\bar{E}| = m$. We construct an instance $G = (V, E, \ell)$ of K13-DEL in the following way: For every vertex $i \in V$ we add two copies i_1 and i_2 and add the three terminals a, b and c. The edge set E consists of edges in E^v and E^e.

$$\begin{aligned}
V &= \{i, i_1, i_2 \mid i \in \bar{V}\} \cup \{a, b, c\} \\
E^v &= \{(i_1, a), (i, b), (i_2, c), (i_1, i), (i, i_2) \mid i \in \bar{V}\} \\
E^e &= \{(i_1, j_2), (j_1, i_2) \mid (i, j) \in \bar{E}\} \\
E &= E^v \cup E^e
\end{aligned}$$

In Figure 9.2 the construction for an edge $(i, j) \in \bar{E}$ is displayed. The edge weights are set as follows:

$$\begin{aligned}
\ell(i_1, a) = \ell(i, b) = \ell(i_2, c) &= 2m(n+1) && \text{for } i \in \bar{V} \\
\ell(i_1, i) = \ell(i, i_2) &= 2m && \text{for } i \in \bar{V} \\
\ell(i_1, j_2) = \ell(i_2, j_1) &= 1 && \text{for } (i, j) \in \bar{E}
\end{aligned}$$

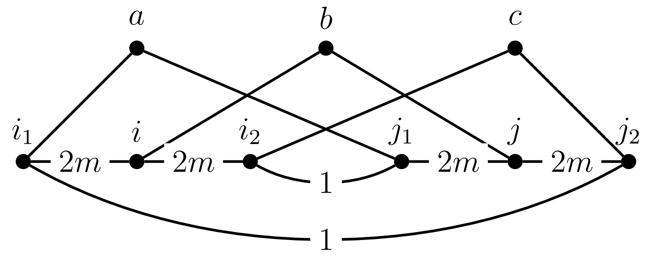


Figure 9.2: Construction of the K13-DEL instance for an edge $(i, j) \in \bar{E}$.

Given the Max-Cut instance $\bar{G}$ the K13-DEL instance $G = (V, E, \ell)$ can be computed in polynomial time. We claim that $\bar{G}$ has a cut of size K or greater if and only if G has a feasible solution with weight of at most $2m(n + 1) - K$.

Let $(\bar{V}_1, \bar{V}_2)$ be a cut of size K or greater. Then we set

$$\begin{aligned} E' = &\{(i_1, i) \mid i \in \bar{V}_1\} \cup \{(i, i_2) \mid i \in \bar{V}_2\} \cup \\ &\{(i_2, j_1) \mid (i, j) \in \bar{E}, i \in \bar{V}_1, j \in \bar{V}_2\} \cup \\ &\{(i_2, j_1), (i_1, j_2) \mid (i, j) \in \bar{E} \text{ and } i, j \in \bar{V}_1 \text{ or } i, j \in \bar{V}_2\} \end{aligned}$$

If $i \in \bar{V}_1$ then $(i_1, i) \in E'$ and $(i_2, j_1) \in E'$ for all $(i, j) \in \bar{E}$. Therefore, the vertices i and i_2 can only reach b and c. The only possibility for vertex i_1 to reach terminal b is to use an edge (i_1, j_2). By construction (i_1, j_2) (for some $(i, j) \in \bar{E}$) is not deleted only if $j \in \bar{V}_2$ and hence $(j, j_2) \in E'$. Therefore, there are only two kinds of edges incident to j_2: Either (j_2, c) or (j_2, k_1) for some $(j, k) \in \bar{E}$ with $k \in \bar{V}_1$. Hence, i_1 can only reach a or c and non-terminal vertices j_2 for $j \in \bar{V}_2$ and k_1 for $k \in \bar{V}_1$. An analogous argument holds for $i \in \bar{V}_2$. Therefore, there is no vertex that reaches all three terminals directly. Hence, G' is feasible. Furthermore, we conclude that

$$\ell(E') = 2mn + K + 2(m - K) = 2m(n + 1) - K$$

holds.

Now we assume that $G' = (V, E \setminus E')$ is feasible for K13-DEL and $\ell(E') \leq 2m(n+1) - K$. Observe that all edges that are incident to terminals have weight $2m(n+1)$. Therefore, we conclude that E' does not contain any of these edges. Moreover, E' contains at least one edge of $\{(i, i_1), (i, i_2)\}$ for all $i \in \bar{V}$ because otherwise G' would not be feasible. Assume that there exists a vertex $i \in \bar{V}$ such that both edges (i, i_1) and (i, i_2) are in E'. Then the weight of E' is bounded by

$$\ell(E') \geq (n - 1)2m + 2(2m) = 2m(n + 1) > 2m(n + 1) - K.$$

This contradicts the assumption of the weight of E'. Hence, we conclude that for every $i \in \bar{V}$ either $(i, i_1) \in E'$ or $(i, i_2) \in E'$ but not both.

If $(i, i_1) \in E'$ and $(i, i_2) \notin E'$ then $(i_2, j_1) \in E'$ for all $(i, j) \in \bar{E}$ since otherwise i_2 would reach all terminals. And if $(i, i_2) \in E'$ and $(i, i_1) \notin E'$ then $(i_1, j_2) \in E'$ for all $(i, j) \in \bar{E}$. Hence, let $(i, j) \in \bar{E}$, and let $(i, i_1) \in E'$ and $(j, j_2) \in E'$. Then the edges in E^e corresponding to (i, j) contribute at least 2 to the value of E'. The same holds if $(i, i_1), (j, j_2) \in E'$. However, if $(i, i_1), (j, j_1) \in E'$ (resp. $(i, i_2), (j, j_2) \in E'$) then at least $(i_2, j_1) \in E'$ (resp. $(i_1, j_2) \in E'$) and therefore the edges in E^e corresponding to (i, j) contribute at least 1 to the value of E'. We set

$$\begin{aligned} \bar{V}_1 &= \{i \in V \mid (i, i_1) \in E'\} \\ \bar{V}_2 &= \{i \in V \mid (i, i_2) \in E'\}. \end{aligned}$$

Let $\bar{C}$ be the size of cut $(\bar{V}_1, \bar{V}_2)$. Then we have

$$\begin{aligned}
2m(n+1)-K &\geq \ell(E') \\
&\geq 2mn + |\{(i,j)\in \bar{E} \mid i,j\in \bar{V_1} \text{ or } i,j\in \bar{V_2}\}| + \\
&\quad 2|\{(i,j)\in \bar{E} \mid i\in \bar{V_1}, j\in \bar{V_2}\}| \\
&= 2mn + \bar{C} + 2(m-\bar{C}) = 2m(n+1)-\bar{C}
\end{aligned}$$

We conclude that $\bar{C} \geq K$ holds.

K13-DEL on Planar Graphs

In this section we develop an $O(n^3)$ time algorithm for K13-DEL on planar graphs. Throughout this section an instance of K13-DEL is given by an edge-weighted planar graph $G = (V, E, \ell)$ and three terminals a, b and c. Let $X = \{a, b, c\}$. Observe that we can assume without loss of generality that G does not contain any terminal-connecting edges. If G contains a terminal-connecting edge, e. g., $(a, b) \in E$, then a subgraph G' of G is feasible if and only if $G' + (a, b)$ is feasible. Hence, $(a, b) \notin E'$ holds for any optimal solution and we can delete (a, b) from G in advance and solve K13-DEL on the reduced graph.

Let us start with a structural investigation of a feasible solution, i.e., a graph $G' = (V, E')$ that does not contain any $K_{1,3}[X]$-minor of the form explained above. Observe that G' may consist of several connected components. We distinguish the following three cases:

1. a, b and c are the only vertices of one connected subgraph. We call this situation type I.
2. a, b and c are not in one connected component. This constellation is called type II.
3. a, b and c are in one connected component that contains at least one further vertex: Let $\hat{G}$ be the connected component of G' containing a, b and c. Consider the dual graph $\hat{G}_D$ of $\hat{G}$ such that the outer face does not correspond to a terminal. Such a dual graph can always be found since there are no edges connecting the terminal vertices. Let A, B, C be the faces associated with the terminals a, b, c.

   Now we delete all edges of $\hat{G}_D$ that are not part of the boundaries of A, B or C. The resulting graph H_D contains at least four faces, face A, B, C and one outer face that corresponds to an accumulation of some non-terminal vertices. Assume that the boundary of the outer face in H_D consists of boundary edges of A, B and C. See Figure 9.7.

   Then there exists a vertex v in $\hat{G}$ that can reach a, b and c without using a, b or c as intermediate nodes on its paths since there are cut edges in H_D connecting face V with the faces A, B and C. This leads to a contradiction to the feasibility of G'. Hence, we know that the outer face of H_D is bounded by edges of boundaries of at most two different faces A, B or C. Assume without loss of generality that edges of the boundaries of A and B bound the outer face.

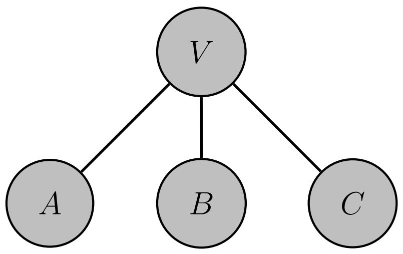


Figure 9.3: The faces A, B and C are adjacent to the outer face.

(a) The boundaries of A and B have at most one vertex in common.
Then the face C is surrounded by the boundary of A or B. See Figure 9.4. This constellation is called type III.

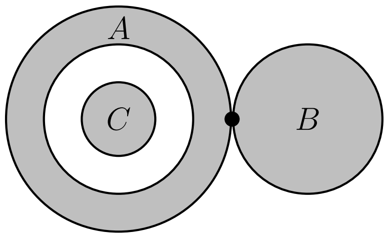


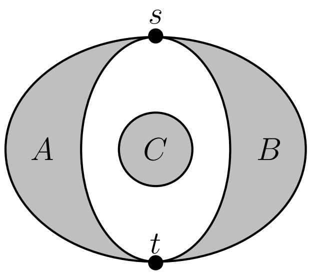
Figure 9.4: Face A surrounds face C and separates C with B.

(b) The boundaries of A and B have at least two vertices in common.
Face C is then surrounded by a part of edges of the boundary of A and a part of the edges of the boundary of B. Hence, there exists a cycle in H_D consisting of two paths, one path from a vertex s to a vertex t containing only edges of the boundary of A and one path from t to s containing only edges from the boundary of B. See Figure 9.5(a).

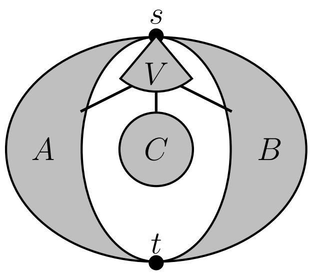
If the boundary of C does not contain vertex s then there exists a face V in H_D such that there exist paths of cut edges from V to A, B and C without crossing the faces A, B or C. See Figure 9.5(b).

This leads to a contradiction to the feasibility of G'. The same argument holds for vertex t. Hence, there are two vertices s and t that lie on the boundaries of A, B and C. See Figure 9.20. This constellation is called type IV.

We have shown that if G' is feasible then it contains one of the types I, II, III or IV. On the other hand, if a graph is of one of these types, it is feasible. Translating type III and IV into the primal graph immediately yields the following structural theorem.



(a) Face A and B have at least two common vertices and C is surrounded by boundary edges of A and B.



(b) Face V can reach A, B and C without crossing one of the terminal faces.

Figure 9.5: Illustration of case 3(b).

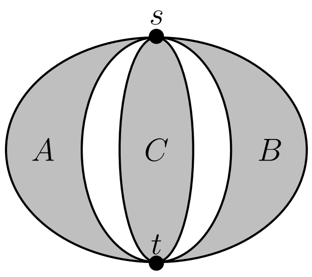


Figure 9.6: Constellation of type IV.

Theorem 9.4. *Given a planar graph G' and three terminal vertices a, b and c. Let $X = \{a, b, c\}$. Then G' does not contain a $K_{1,3}[X]$-minor if and only if*

1. *a, b and c are the only vertices of a connected component, or*
2. *a, b and c are not in the same connected component, or*
3. *at least one terminal is a cut vertex, or*
4. *there exists an embedding such that the three terminals lie on two common boundaries.*

Observe that item 4 may be rewritten as "each pair of a, b, c is a split pair in G'". Hence, if one planar embedding satisfy item 4 then all planar embeddings do.

If graph G' is connected we get the following corollary.

Corollary 9.3. *Given a planar connected graph G' and three terminal vertices a, b and c. Let $X = \{a, b, c\}$. Then G' does not contain a $K_{1,3}[X]$-minor if and only if either at least one terminal is a cut vertex or there exists an embedding such that the three terminals lie on two common boundaries.*

Based on the characterization of $K_{1,3}[X]$-free graphs of Theorem 9.4 we are looking for an edge removal strategy such that the resulting graph satisfies one of the properties of Theorem 9.4. An algorithm solving each type at minimum cost and taking the cheapest solution among the four types would solve the problem. An example is visualized in Figure 9.7, where a graph and its subgraphs G' that correspond to types II-IV are presented. By assuming that each edge has cost 1, type III would be optimal.

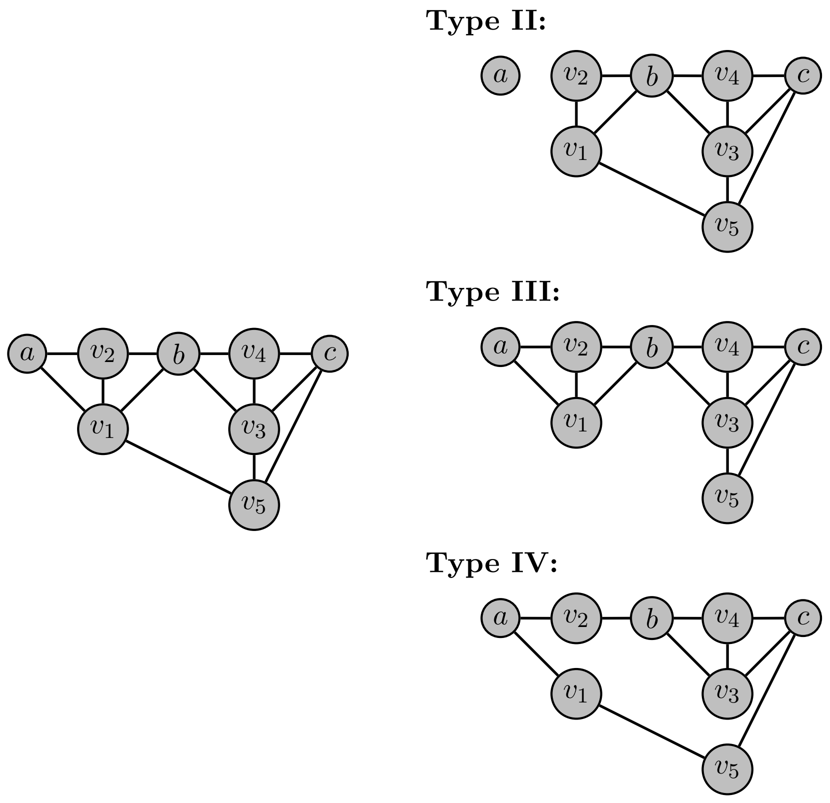


Figure 9.7: A graph with $K_{1,3}[X]$-minors on the left-hand side. On the right-hand side the corresponding subgraphs G' that depend on the types II-IV are visualized. Type I cannot be achieved and therefore is omitted.

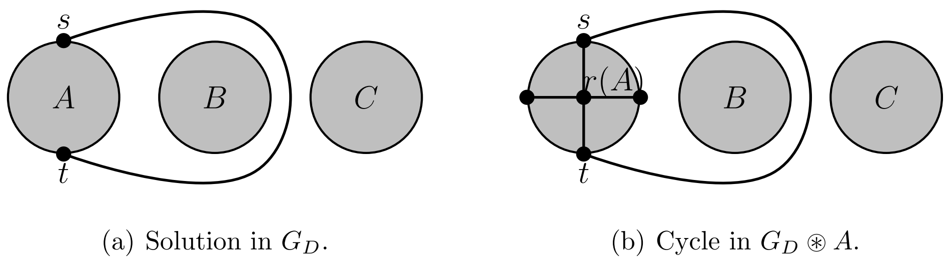
Let us analyze the four types:

Every solution of type I contains all edges that are incident to one terminal and one non-terminal vertex. Since we assume that there are no terminal-connecting edges we conclude that every solution E' of type I yields a pairwise separation of the terminal nodes. Hence, E' is also feasible for type II. Therefore, an optimal solution of type I can not dominate an optimal solution of type II.

A graph is of type II if there exists at least one terminal that is separated from the other two terminals. The problem of finding a type II constellation at minimum cost boils down to three min-cut problems.

Consider type III: Observe that the deletion of an edge in the primal graph leads to the contraction of the edge in the dual graph. In order to achieve type III we have to find a path starting on the boundary of the first terminal face, surrounding the face of the second terminal and ending on the boundary of the first terminal face in such a way that the second and third faces are separated. See Figure 9.8(a).

Let s and t be starting and end vertices of the path on the boundary of A in G_D. We are interested in a path starting at s and ending at t that separates face B from C. Every path of this kind corresponds to a cycle in $G_D \circledast A$ that contains vertex $r(A)$. See Figure 9.8(b).



(a) Solution in G_D. (b) Cycle in $G_D \circledast A$.

Figure 9.8: Solution of type III.

If we set $\ell(e) = 0$ for all edges e of the inserted k-star then the total length of a minimum cycle X_A in $G_D \circledast A$ that separates B from C and contains $r(A)$ is equal to a shortest path that starts and ends on the boundary of A and separates B from C. Consider the following relaxed problem:

$$\begin{array}{lll}(\text{RMC}) & \min & \ell(X) \\ & \text{s.t.} & \exists Z \in \{A, B, C\} \text{ such that } X \text{ is a cycle in } G_D \circledast Z \\ & & \text{that separates the terminals in } \{A, B, C\} \setminus \{Z\}\end{array}$$

We show that the set of feasible solutions of (RMC) is equal to the set of feasible solutions of type II and III.

Lemma 9.2. *An optimal solution of (RMC) is optimal among all type II and type III solutions.*

Proof. Let X be a feasible solution of (RMC) that separates A from B in $G_D \circledast C$. If X contains $r(C)$ then X is a feasible solution of type III. Otherwise, if X does not contain $r(C)$, then X is a feasible solution of type II since it separates A from B in G_D. Together with the same argumentation for A and B yields that every feasible solution of (RMC) is of type II or III. On the other hand, let X_A be a feasible solution of type III that separates B from C and contains $r(A)$. Then X_A is feasible for (RMC) for $Z = A$. Analogous arguments hold for B and C. And finally, if X_{AB} is a solution of type II that separates A and B from C in G_D, then X_{AB} separates B from C in $G_D \circledast A$. Since analogous arguments hold for X_{AC} and X_{BC} we conclude that every solution of type II or III is feasible for (RMC). □

Lemma 9.2 implies an algorithm that simultaneously solves type II and type III to optimality. See Algorithm 30 for a pseudo-code description.

Algorithm 30: Algorithm for type II and type III: Output is an optimal solution among all solutions of type II and type III.

```
forall the $Z \in \{A, B, C\}$ do
    Determine a minimum cut in the primal graph of $G_D \circledast Z$ that separates the terminals in $\{A, B, C\} \setminus \{Z\}$.;
    Let $X_Z^*$ denote the determined minimum cut;
$\ell(X^*) \Leftarrow \min\{\ell(X_A^*), \ell(X_B^*), \ell(X_C^*)\}$;
Return: $X^*$;
```

In Algorithm 30 we have to solve three minimum st-cut problems in planar graphs. Hassin and Johnson [46] suggested an $O(n \log^2 n)$ time algorithm for the maximum flow problem in planar graphs. Using Frederickson's shortest path algorithm for planar graphs [31] within the algorithm of Hassin and Johnson yields an $O(n \log n)$ time algorithm for the maximum flow problem and hence for the minimum cut problem in planar graphs. Therefore, an optimal solution of type II and III can be found in time

$$O(n \log n).$$

Finally, we have to show how to solve type IV. To achieve type IV, we have to find two vertex-disjoint connected subgraphs S and T each of them containing at least one vertex of the boundary of A, B and C. Let us fix two vertices of each boundary, i. e., $a_1, a_2 \in \mathrm{bd}(A)$, $b_1, b_2 \in \mathrm{bd}(B)$ and $c_1, c_2 \in \mathrm{bd}(C)$. Clearly S and T are trees in an optimal solution. See Figure 9.25(b).

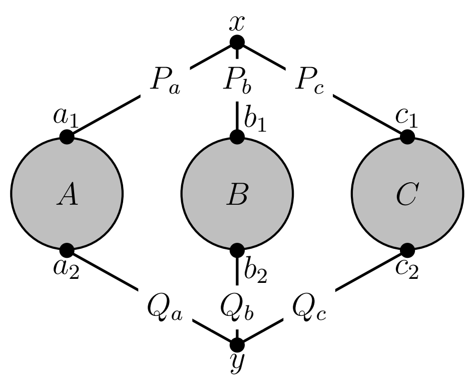


Figure 9.9: Solution of type IV.

Hence, this leads to the vertex disjoint Steiner Tree packing problems, where $\{a_1, b_1, c_1\}$ is one net of terminals and $\{a_2, b_2, c_2\}$ is the second net of terminals. The vertex disjoint Steiner Tree packing problem is in general $\mathcal{NP}$-hard on planar graphs [65]. For several special cases the problem becomes solvable in polynomial time, e. g., if each net has cardinality two and the number of nets is fixed (Robertson and Seymour [71]) or if the terminals lie only on one or two face boundaries (Robertson and Seymour [73], Suzuki, Akama and Nishizeki [79] and Liao and Sarrafzadeh [62]). For a survey on this problem the reader is referred to Wagner [83].

Our approach to this special vertex disjoint Steiner Tree packing problem is to solve the problem to optimality only if the choice of a_1, a_2, b_1, b_2, c_1 and c_2 is a "potential optimal choice". If we can guarantee that an optimal solution for a fixed choice of boundary vertices is dominated by a solution of type III or a solution of type IV with another choice of a_i, b_i and c_i we do not solve the corresponding vertex disjoint Steiner Tree packing problem to optimality. Moreover we know that tree S contains at most one vertex of degree 3 since it has to connect three terminals. The same observation holds for tree T. If S contains no vertex of degree 3 then S is a path. If S is a path then there exists exactly one terminal, i. e., a vertex on the boundary of a terminal face, that is not a leave. This vertex is called degenerated degree-3-vertex.

Let $x, y \in V_D$ be the two degenerated degree 3-vertices of S and T. Then $S \cup T$ is a disjoint union of the paths P_z and Q_z where P_z is a path from x to a boundary vertex of terminal $z \in \{a, b, c\}$ and Q_z is a path from a boundary vertex of terminal $z \in \{a, b, c\}$ to y.

Lemma 9.3. *Assume that an optimal solution of problem K13-DEL is of type IV. Let $S \cup T$ be the corresponding optimal set of edges and let x and y be the two (degenerated) degree 3 vertices. Furthermore, let*

$$\min\{\ell(P_a) + \ell(Q_a), \ell(P_b) + \ell(Q_b), \ell(P_c) + \ell(Q_c)\} = \ell(P_f) + \ell(Q_f)$$

for $f \in \{a, b, c\}$. Then $(P_f, r(F), Q_r)$ is a shortest path from x to y in $G_D \circledast F$.

Proof. The proof is by contradiction. Assume that there exists a path P^* from x to y in $G_D \circledast F$ with

$$\min\{\ell(P_a) + \ell(Q_a), \ell(P_b) + \ell(Q_b), \ell(P_c) + \ell(Q_c)\} > \ell(P^*). \tag{9.1}$$

Furthermore, let i and j be two vertices on $P^* \cap (S \cup T)$. Then $P^*(i, j)$ denotes the subpath of P^* from i to j and $P(i, j)$ denotes the unique subpath of $S \cup T$ from i to j. Observe that we can assume without loss of generality that $\ell(P^*(i, j)) < \ell(P(i, j))$ holds for all pairs of vertices $i, j \in P^* \cap (S \cup T)$.

Now we consider a special pair of vertices: Let i be the first vertex on P^* on its way from x to y such that the edge leaving i is not in $S \cup T$ and let j be the first vertex after i on P^* that is on $S \cup T$. Since x is on $P_a \cup P_b \cup P_c$ we conclude that i is on $P_a \cup P_b \cup P_c$ or on Q_f. Now we distinguish the following cases:

1. Both vertices i and j are on the same path P_a, P_b or P_c. See Figure 9.10:

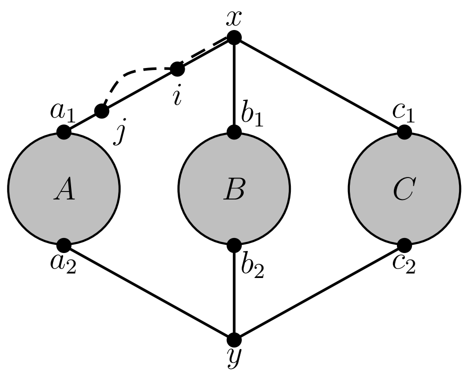


Figure 9.10: Vertex i and j are on P_a.

We set

$$S' = S \cup P^*(i,j) \setminus P(i,j)$$
$$T' = T$$

Since $S' \cup T'$ is a feasible solution and

$$\ell(P^*(i,j)) < \ell(P(i,j))$$

holds, we have a contradiction to the optimality of $S \cup T$.

2. Both vertices i and j are on a path P_z for $z \in \{a,b,c\}$ but not on the same; see Figure 9.11: We can assume without loss of generality that $i \neq x$ because otherwise we get the first case.

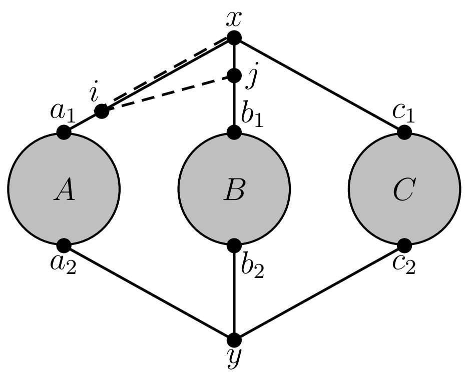


Figure 9.11: Vertex i is on P_a and j is on P_b.

Observe that the following chain of inequalities holds

$$\ell(P^*(i,j)) < \ell(P^*(x,j)) < \ell(P(x,j))$$

We set

$$S' = S \cup P^*(i,j) \setminus P(x,j)$$
$$T' = T$$

S' is a tree that covers a_1, b_1 and c_1 where vertex i is the degree 3-vertex. Furthermore $S' \cup T'$ dominates $S \cup T$. Hence, vertex x was the wrong choice for the degree 3-vertex. This leads to a contradiction to the optimality of $S \cup T$ and x, y as degree 3-vertices.

3. Vertex $i \neq x$ is on a path P_z and j is on some $Q_{z'}$ for $z \neq z'$; see Figure 9.12:

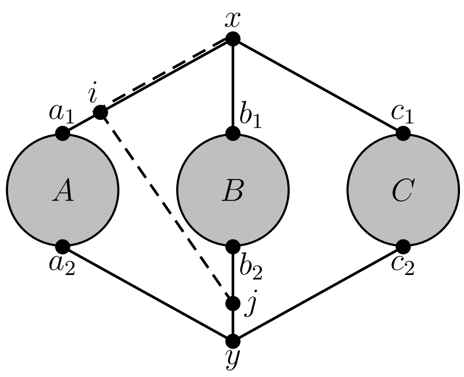


Figure 9.12: Vertex i is on P_a and j is on Q_b.

Then the following chain of inequalities holds:

$$\begin{aligned}\ell(P^*(x,j)) &\leq \ell(P^*(x,y)) \\ &< \ell(P_f) + \ell(Q_f) \\ &\leq \ell(P_z) + \ell(Q_z)\end{aligned}$$

We conclude that

$$\ell(P^*(x,j)) < \ell(P_z) + \ell(Q_z)$$

holds. We set

$$S' \cup T' = S \cup T \cup P^*(x,j) \setminus (P_z + Q_z)\,.$$

Observe that $S' \cup T'$ dominates $S \cup T$. Now we show that $S' \cup T'$ is a solution of type III.

In order to simplify the notation let $z = a$ and $z' = b$. See Figure 9.12. Then $S' \cup T'$ contains a path $\hat{P}$ consisting of the edges of P_c, $P_a(x,i)$, $P^*(i,j)$, $Q_b(j,y)$ and Q_c such that $\hat{P}$ is a path from c_1 to c_2 separating the faces A and B. Hence, $S' \cup T'$ is a solution of type III which dominates $S \cup T$.

4. Vertex i is on P_z and vertex j is on Q_z:

   Let k be the next vertex after j on P^* such that k is on $S \cup T$ but the edge arriving at k is not in $S \cup T$.

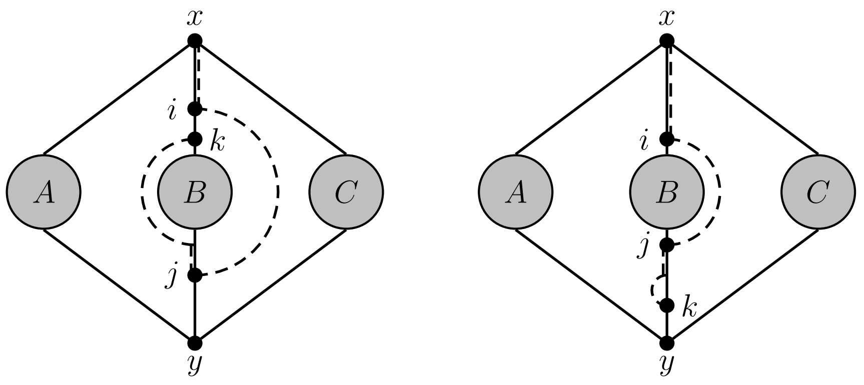
(a) If k is on P_z (see Figure 9.13(a)) we conclude

$$\ell(P^*(i,j)) < \ell(P^*(i,k)) < \ell(P(i,k)) = \ell(P_z(i,k)).$$

Set

$$S' \cup T' = S \cup T \cup P^*(i,j) \setminus P_z(i,k).$$

$S' \cup T'$ is a solution of type III that dominates $S \cup T$.



(a) Case 4(a) for $z = b$: Vertex i is on P_b, j is on Q_b and k is on P_b.

(b) Case 4(b) for $z = b$: Vertex i is on P_b and j and k are on Q_b.

Figure 9.13: Proof of Lemma 9.3: Subcases of 4.

(b) If k is on Q_z (see Figure 9.13(b)) we know that

$$\ell(P^*(j,k)) < \ell(P(j,k)) = \ell(Q_z(j,k)) \tag{9.2}$$

We set

$$S' \cup T' = S \cup T \cup P^*(j,k) \setminus Q_z(j,k)$$

$S' \cup T'$ is of type IV and dominates $S \cup T$.

(c) If k is on $P_{z'}$ for $z' \in \{a,b,c\} \setminus \{z\}$ (see Figures 9.14(a) and 9.14(b)) then the following two chains of inequalities hold:

$$\ell(P^*(x,k)) < \ell(P(x,k)) = \ell(P_{z'}(x,k))$$

and

$$\ell(P^*(x,k)) \leq \ell(P^*(x,y)) < \ell(P_f) + \ell(Q_f) \leq \ell(P_z) + \ell(Q_z)$$

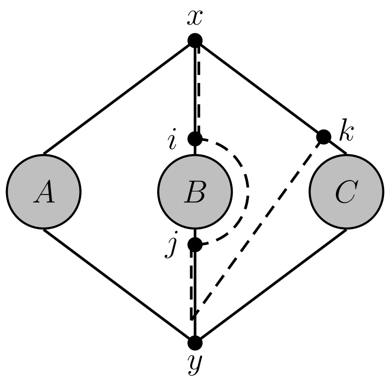
If $P^*(i,j)$ lies in the region bounded by $(P_z, Z, Q_z, Q_{z'}, Z', P_{z'})$ (Figure 9.14(a)) we set

$$S' \cup T' = S \cup T \cup P^*(x,k) \setminus P_{z'}(x,k)$$

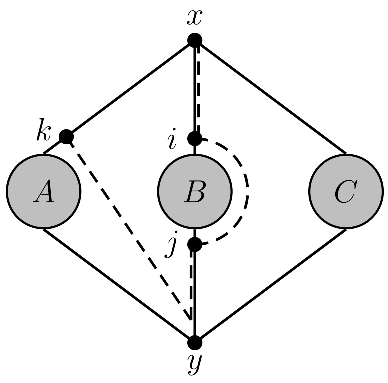
and get a solution of type III otherwise (Figure 9.14(b)) we set

$$S' \cup T' = S \cup T \cup P^*(x,k) \setminus (P_z, Q_z)$$

and get a solution of type II. In both cases the solution $S' \cup T'$ dominates $S \cup T$ and therefore leads to a contradiction to the optimality of $S \cup T$.



(a) Case 4(c) for $z = b$ and $z' = c$: $P^*(i,j)$ lies in the region bounded by $(P_z, Z, Q_z, Q_{z'}, Z', P_{z'})$.



(b) Case 4(c) for $z = b$ and $z' = a$: $P^*(i,j)$ lies not in the region bounded by $(P_z, Z, Q_z, Q_{z'}, Z', P_{z'})$.

Figure 9.14: Proof of Lemma 9.3: Subcases of 4.

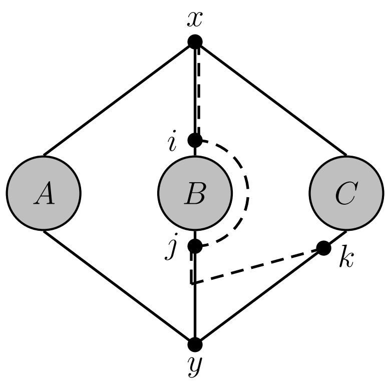
(d) Finally, if k is on $Q_{z'}$ (see Figures 9.15(a) and 9.15(b)) we know that

$$\ell(P^*(x,k)) \leq \ell(P^*(x,y)) < \ell(P_z) + \ell(Q_z). \tag{9.3}$$

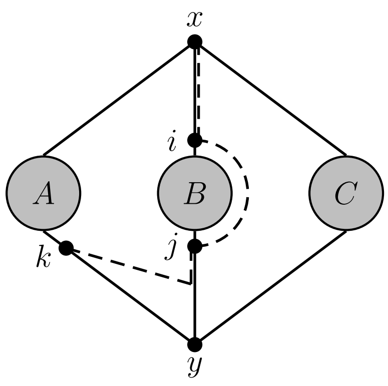
We set

$$S' \cup T' = S \cup T \cup P^*(x,k) \setminus (P_z + Q_z)$$

and get a solution of type III that dominates $S \cup T$.



(a) Case 4(c) for $z = b$ and $z' = c$.



(b) Case 4(c) for $z = b$ and $z' = a$.

Figure 9.15: Proof of Lemma 9.3: Subcases of 4.

This completes the proof of case 4.

5. Vertex i is on Q_f:

   Since P^* is a shortest path from x to y we conclude that P^* is also a shortest path from y to x. Hence, we consider the following pair of vertices: Let i' be the first vertex on P^* on its way from y to x such that the edge leaving i' is not in $S \cup T$ and let j' be the first vertex after i' on P^* on its way from y to x that is on $S \cup T$. Then we can use the same arguments as above by replacing i by i' and j by j' and changing the roles of the paths P_z and Q_z; for $z \in \{a, b, c\}$.

This completes the proof. □

Lemma 9.3 implies the following main theorem.

Theorem 9.5. *Assume that an optimal solution of problem K13-DEL is of type IV. Let $S \cup T = P_a \cup P_b \cup P_c \cup Q_a \cup Q_b \cup Q_c$ be the corresponding optimal set of edges and let x and y be the two degenerated degree 3 vertices. Moreover, let*

$$\ell(P_f) + \ell(Q_f) \leq \ell(P_g) + \ell(Q_g) \leq \ell(P_h) + \ell(Q_h)$$

for $\{f, g, h\} = \{a, b, c\}$. Then the following holds:

- *$(P_f, r(F), Q_f)$ is a shortest path from x to y in $G_D \circledast F$.*
- *$(P_g, r(G), Q_g)$ is a shortest path from x to y in $G_D \circledast G/\,(P_f, Q_f)$.*
- *$(P_h, r(H), Q_h)$ is a shortest path from x to y in $G_D \circledast H/\,(P_f, Q_f, P_g, Q_g)$.*

Proof. The correctness of the shortest path condition on $(P_f, r(F), Q_f)$ was already proved in Lemma 9.3.

Now we contract P_f and Q_f. Clearly, $S \cup T \setminus (P_f, Q_f)$ is an optimal solution of the contracted instance. Assume that there exists a path P^* from x to y in $G_D \circledast G/\,(P_f, Q_f)$ with $\ell(P^*) < \ell(P_g) + \ell(Q_g)$. We use the technique of the proof of Lemma 9.3 where P_f and Q_f of the lemma play the role of P_g and Q_g, respectively, here. Since P^* does not contain any vertex of $P_f \cup Q_f$ except for x and y all inequalities of the proof of Lemma 9.3 hold for the contracted instance, too. Hence, we can find a contradiction to the optimality of $S \cup T \setminus (P_f, Q_f)$. This proves the shortest path condition on $(P_g, r(G), P_g)$.

Finally, we have to prove the shortest path condition on $(P_h, r(H), Q_h)$. For any xy-path P^* in $G_D \circledast H/\,(P_f, Q_f, P_g, Q_g)$ the solution $P_f \cup Q_f \cup P_g \cup Q_g \cup P^*$ is a feasible solution: If P^* contains $r(H)$ the corresponding solution is of type IV. Otherwise, if P^* does not contain $r(H)$, the resulting solution is of type III. Therefore, $(P_h, r(H), Q_h)$ has to be a shortest path in an optimal solution. This completes the proof. □

Observe that Theorem 9.5 implies that the xy-paths of an optimal solution are shortest paths in appropriate graphs. However, the shortest path condition is not only necessary but also sufficient for the optimality of a solution of K13-DEL.

Lemma 9.4. *Assume that an optimal solution $S \cup T$ of problem K13-DEL is of type IV, and let x and y be the degenerated degree 3 vertices. Furthermore, let P'_z be a shortest path from x to $r(Z)$, and let Q'_z be a shortest path from $r(Z)$ to y in $G_D \circledast Z$ for $z \in \{a, b, c\}$. Then there exists a terminal $f \in \{a, b, c\}$ and an optimal solution that contains P'_f and Q'_f.*

Proof. Let $S \cup T$ be an optimal solution that contains the maximal number of edges in $P'_f \cup Q'_f$. We know that $S \cup T$ is of the form $P_a \cup P_b \cup P_c \cup Q_a \cup Q_b \cup Q_c$. Let

$$\min\{\ell(P_a) + \ell(Q_a), \ell(P_b) + \ell(Q_b), \ell(P_c) + \ell(Q_c)\} = \ell(P_f) + \ell(Q_f)$$

Observe that (P'_f, Q'_f) satisfies the necessary optimality condition of Theorem 9.5. However, assume that P_f and Q_f differs from P'_f and Q'_f.

Let i be the first vertex on $R' = (P'_f, r(F), Q'_f)$ on its way from x to y such that the edge leaving i is not in $S \cup T$, and let j be the first vertex on R' on its way from x to y after i that is on $S \cup T$. We distinguish the five cases of the proof of Lemma 9.3 for $P^* = R'$ and apply the exchange strategies suggested therein. Since $\ell(R') = \ell(P_f) + \ell(Q_f)$ holds, some inequalities of the proof of Lemma 9.3 do not hold strict if P^* is replaced by R'. However, if we end up in a case where the exchange procedure produces a solution that strictly dominates the previous one we get a contradiction to the optimality of $S \cup T$. Otherwise, if the objective value of the modified solution equals the previous one, we get an optimal solution that contains at least one more edge of R'. This contradicts the assumption that $S \cup T$ contains a maximal number of edges of R'. □

Observe that analogous results to that of Lemma 9.4 hold for the paths $(P_g, r(G), Q_g)$ and $(P_h, r(H), Q_h)$.

Combining the results of Theorem 9.5 and Lemma 9.4 immediately leads to an algorithm that solves type IV: Using enumeration we fix one order (f, g, h) of the faces. From Lemma 9.4 we know that there exists an optimal solution that contains every pair of shortest paths P'_f and Q'_f. According to Theorem 9.5 we contract P'_f and Q'_f and determine a pair of shortest paths P'_g and Q'_g. Lemma 9.4 applied to the contracted instance guarantees that P'_g and Q'_g is contained in an optimal solution of the contracted instance. The same procedure is repeated for terminal h. See Algorithm 31 for a pseudo-code description.

It remains to bound the running time of Algorithm 31. The degree 3 vertices x and y can be chosen in

$$\binom{n}{2} = O(n^2)$$

different ways. Furthermore, there are $3! = 6$ possible orderings of the terminal vertices. And finally, for every ordering we have to solve a sequence of three shortest path problems. Frederickson [31] showed that a shortest path in an undirected, planar graph can be found in $O(n\sqrt{\log n})$ time. Hence, the running time of Algorithm 31 is bounded by

$$O(n^3\sqrt{\log n}).$$

Algorithm 31: Determination of an optimal solution of type IV: Output is an optimal solution of type IV.

```
forall the pairs of vertices $x, y \in V$ do
    forall the orderings $(f, g, h)$ with $\{f, g, h\} = \{a, b, c\}$ do
        Determine shortest paths $P_f$ from $x$ to $r(F)$ and $Q_f$ from $r(F)$ to $y$;
        Contract $P_f$ and $Q_f$;
        Determine shortest paths $P_g$ from $x$ to $r(G)$ and $Q_g$ from $r(G)$ to $y$;
        Contract $P_g$ and $Q_g$;
        Determine shortest paths $P_h$ from $x$ to $r(H)$ and $Q_h$ from $r(H)$ to $y$;
        $\mathrm{Sol}(f, g, h) \Leftarrow P_f \cup Q_f \cup P_g \cup Q_g \cup P_h \cup Q_h$;
        $\mathrm{val}(f, g, h) \Leftarrow \ell(\mathrm{Sol}(f, g, h))$;
    $\mathrm{val}(x, y) \Leftarrow \min\{\mathrm{val}(f, g, h) \mid \{f, g, h\} = \{a, b, c\}$ and $\mathrm{Sol}(f, g, h)$ is feasible$\} =$
    $\mathrm{val}(f^*, g^*, h^*)$;
    $\mathrm{Sol}(x, y) \Leftarrow \mathrm{Sol}(f^*, g^*, h^*)$;
$\mathrm{val} \Leftarrow \min\{\mathrm{val}(x, y) \mid x, y \in V\} = \mathrm{val}(x^*, y^*)$;
Return: Sol $\Leftarrow \mathrm{Sol}(x^*, y^*)$;
```

Henzinger et al. [47] showed that a shortest path in a planar graph can be found in linear time. Hence, the running time of Algorithm 31 can be bounded by

$$O(n^3).$$

Putting all together yields a polynomial time algorithm for K13-DEL:

Theorem 9.6. *Problem K13-DEL in a planar graph $G = (V, E)$ can be solved in $O(n^3)$ time where $n = |V|$.*

Proof. Algorithm 30 determines an optimal solution of type II and III and Algorithm 31 determines an optimal solution of type IV. The solution with smaller objective value is then an optimal solution of K13-DEL.

The correctness of this algorithm follows from Theorem 9.5 and Lemma 9.4 and the running time follows from the complexity analysis of Algorithm 30 and Algorithm 31. □

9.2.3 The Rooted $K_{2,3}$-deletion Problem

In this section we consider the K23-DEL problem on general graphs and on planar graphs.

K23-DEL on General Graphs

In order to prove $\mathcal{NP}$-hardness of K23-DEL on general graphs we use the construction of Section 9.2.2 and add a rooted $K_{1,3}[X]$, with $X = \{a, b, c\}$, whose edges have sufficiently large weight. This modified construction immediately leads to the following corollary:

Corollary 9.4. *The weighted rooted $K_{2,3}$-deletion problem is in general $\mathcal{NP}$-hard.*

Obviously, we can state the following corollary.

Corollary 9.5. *The weighted rooted $K_{i,3}$-deletion problem for a fixed $i \in \mathbb{N}$ is $\mathcal{NP}$-hard in general.*

K23-DEL on Planar Graphs

First we characterize detailed a planar graph G that has no $K_{2,3}[X]$-minor with respect to three given vertices a, b, c. Let $X = \{a,b,c\}$. Then we show how we can solve a partial problem of finding the minimum number of edges in a given planar graph G in polynomial time in order to delete all $K_{2,3}[X]$-minors. For the remaining cases we conjecture that it is also polynomial time solvable.

Again, let us start with a structural investigation of a feasible solution of a $K_{2,3}[X]$-free graph G'. Observe that G' may consist of several connected components.

Obviously, if G has no $K_{1,3}[X]$-minor then it also has no $K_{2,3}[X]$-minor. Therefore, all $K_{1,3}[X]$-free subgraphs of G are feasible solutions (see Section 9.2.2, types I - IV).

Additionally, all $K_{2,3}[X]$-free subgraphs of G with a $K_{1,3}[X]$-minor are feasible solutions. We show that these subgraphs are those that have a planar embedding with the three terminals on a common boundary in this section.

First we give some results by Wagner [81, 84], Hall [81] and Truemper [81] that we use to prove the main result of this section.

Theorem 9.7 (Wagner [81, 84]). *A graph is planar if and only if it has no minor isomorphic to K_5 or $K_{3,3}$.*

Theorem 9.8 (Hall [81]). *A graph has no minor isomorphic to $K_{3,3}$ if and only if it can be obtained from planar graphs and K_5 by means of 0-, 1- and 2-sums.*

Notice that it follows immediately from the last theorem that a triconnected non-planar graph G has no minor isomorphic to $K_{3,3}$ if and only if G is isomorphic to K_5.

Theorem 9.9 (Truemper [81]). *Let G be a 3-connected graph with a $K_{3,3}$ minor. If G has a node u of degree 3, then G has as subgraph a subdivision of $K_{3,3}$ that has u as a corner node.*

Additionally, we need the following lemma for the proof.

Lemma 9.5. *Given a biconnected planar graph $G = (V, E)$ and three terminal vertices a, b, c of V. Let $\hat{G}$ be the modified graph G after adding a 3-star S whose leaves are a, b, c. Then $\hat{G}$ has exactly one triconnected prime $\hat{C}$ in its decomposition into triconnected primes that contains a, b, c. Furthermore, $\hat{C}$ contains S.*

Proof. Obviously, if G is triconnected then $\hat{G}$ is triconnected. Furthermore, there is a unique decomposition into triconnected primes for a biconnected graph.

We assume that G is biconnected but not triconnected. Then G has a split pair s, t of V such that G splits into k triconnected primes C_1 to C_k ($k \in \mathbb{N}$) by removing s, t from G. We state the following cases.

1. a, b, c are contained in the same triconnected prime C_i, $1 \leq i \leq k$ (see Figure 9.16). Then S is added to C_i and we get a triconnected prime $\hat{C}_i$. Since the decomposition into triconnected primes is unique and $G = \hat{G} - S$, $\hat{C}_i$ is equal to exactly one triconnected prime of $\hat{G}$.

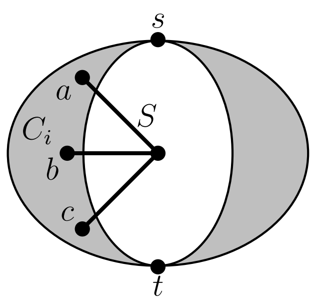


Figure 9.16: Proof of Lemma 9.5, case 1

2. W.l.o.g., a, b belong to the same triconnected prime C_i, and c to an other C_j with $1 \leq i, j \leq k$, $i \neq j$. See Figure 9.17. Then at least one terminal of a, b is not equal to s or t, say a. Furthermore, c cannot be equal to s or t. Therefore, we have at least two vertices that are not equal to s or t, say a and c.

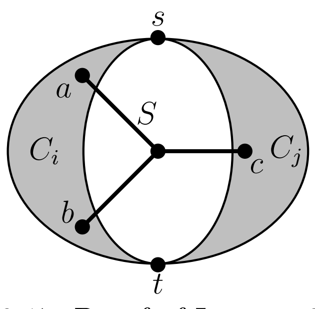


Figure 9.17: Proof of Lemma 9.5, case 2

If we add S, we connect a and c by a path P not using b. Therefore, P connects C_i and C_j without using s or t. Therefore, for each vertex pair $x \in C_i$,$y \in C_j$, there is an additional disjoint path over P. Hence, for each vertex pair x, y there are three disjoint paths in total. By Menger's well-known result, s and t do not remain split pair of C_i and C_j. This implies that C_i and C_j merge into a single triconnected prime, say $\hat{C}_l$ ($l \in \mathbb{N}$), that contains S. Because of uniqueness of the decomposition into triconnected prime and $G = \hat{G} - S$, $\hat{G}$ has exactly one triconnected prime isomorphic to $\hat{C}_l$.

3. a, b, c are belonging to three different triconnected primes C_i, C_j, C_l ($1 \leq i,j,l \leq k$, $i \neq j \neq l$) of G. See Figure 9.18. Therefore, a, b, c are not equal to s or t. If we add S, we connect a, b, c.

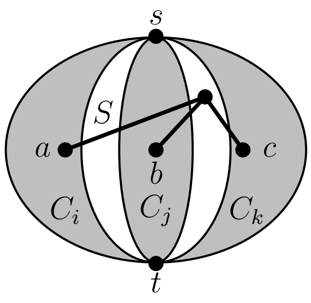


Figure 9.18: Proof of Lemma 9.5, case 3

Therefore, there is a path P_1 connecting C_i and C_j resp. a path P_2 connecting C_j and C_l resp. a path P_3 connecting C_l and C_i. Observe that neither P_1 nor P_2 nor P_3 use s or t. Therefore, there are three disjoint paths that connect a vertex of C_i and a vertex of C_j. The same holds for vertex pairs of C_j and C_k resp. C_i and C_k. By Menger, s and t do not remain split pairs of C_i, C_j, C_l. Hence, C_i, C_j, C_l merge into a single triconnected prime, say $\hat{C}_r$ ($r \in \mathbb{N}$), that contains S. Since the decomposition into triconnected primes is unique and $G = \hat{G} - S$, $\hat{G}$ has exactly one triconnected prime equal to C_r.

□

Now we can prove the following main result of this section.

Theorem 9.10. *Given a biconnected planar graph G and three terminal vertices a, b and c with $X = \{a,b,c\}$. Then G does not contain a $K_{2,3}[X]$-minor if and only if*

1. *G has no $K_{1,3}[X]$-minor or*
2. *there exists a planar embedding such that the three terminals lie on a common boundary.*

Proof. Let G^* be G with a 3-star S connected to a, b and c. By Lemma 9.5, G^* has exactly one triconnected prime, say C^*, in its decomposition into triconnected prime that contains a, b, c. Furthermore, C^* contains S.

1. (a) Assume G has no $K_{1,3}[X]$-minor. Since $K_{2,3}[X]$ contains two disjoint $K_{3,1}[X]$, G has obviously no $K_{2,3}[X]$-minor.

   (b) G has a planar embedding such that a, b, c lie on a comon boundary. Assume G has a $K_{2,3}[X]$-minor.

   Obviously, G^* is planar. Since G has a $K_{2,3}[X]$-minor, G^* has a $K_{3,3}[X]$-minor. Therefore G^* has a $K_{3,3}$ minor. By Theorem 9.7, G^* is non-planar that leads to a contradiction. Therefore, G has no $K_{2,3}[X]$-minor.

2. G is planar. Furthermore, there is no planar embedding of G such that a, b, c lie on the boundary of the same face. Assume that G has no $K_{2,3}[X]$-minor.

   G^* is not planar since G^* has no planar embedding. Since $G = G^* - S$ is planar, G^* has a $K_{3,3}[X]$- or $K_5[X]$-minor. Observe that $K_{3,3}$ and K_5 are triconnected. Since the decomposition into triconnected prime of a biconnected graph is unique, a $K_{3,3}[X]$-resp. $K_5[X]$-minor is contained in the same triconnected prime as a, b, c and S that is C^*. Therefore, C^* is non-planar.

   Assume, G^* has a $K_5[X]$-minor but no $K_{3,3}[X]$-minor. Therefore, G^* has a K_5 minor but no minor isomorphic to $K_{3,3}$. By Wagner's Theorem 9.7 and Hall's Theorem 9.8, C^* is isomorphic to K_5. Since C^* contains S and therefore at least one vertex with degree 3 this leads to a contradiction.

   Therefore, we assume that C^* is not isomorphic to K_5 and not planar. By Wagner's Theorem 9.7, C^* has a minor isomorphic to $K_{3,3}$. Since $G = G^* - S$ is planar, $C^* - S$ is planar. Hence C^* has a $K_{3,3}[X]$-minor. Notice that C^* contains S and therefore at least one vertex u with degree 3. Let u be the vertex of S. By Truemper's Theorem 9.9, C^* has a subgraph a subdivision of $K_{3,3}$ that has u as a corner node. Therefore, C^* has a subgraph a subdivision of $K_{3,3}[X]$, that has u as a corner node. Since S is isomorphic to $K_{1,3}[X]$, $C^* - S = C^* - u$ has a $K_{2,3}[X]$ subdivision. Therefore, G has a $K_{2,3}[X]$ subdivision. This contradicts the assumption that G has no $K_{2,3}[X]$-minor.

□

Theorem 9.10 gives us a characterization of $K_{2,3}[X]$-free biconnected graphs.

First we focus on the characterization of planar biconnected graphs with the property that a planar embedding exists such that the three terminals a, b and c lie on a common boundary.

Lemma 9.6. *Given a biconnected planar graph G' and three terminal vertices a, b and c ($X = \{a,b,c\}$), and let $\{e,f,g\} = \{a,b,c\}$.*

There exists a planar embedding of G' such that the three terminals lie on a common boundary if and only if

1. *for all planar embeddings of G' a, b, c lie on a common boundary, or*

2. *there exist vertices s, t of G such that for all planar embeddings of G':*

   (a) *s, t lie on two common boundaries,*

   (b) *e, f, s, t lie on a common boundary, and*

   (c) *g, s, t lie on a common boundary,*

   or

3. *there exist vertices s, t, u of G' such that for all planar embeddings of G':*

   (a) *s, t, u lie on a common boundary,*

   (b) *e, s, t lie on a common boundary,*

   (c) *f, t, u lie on a common boundary, and*

   (d) *g, s, u lie on a common boundary.*

   (e) *each pair of s, t, u lie on two common boundaries.*

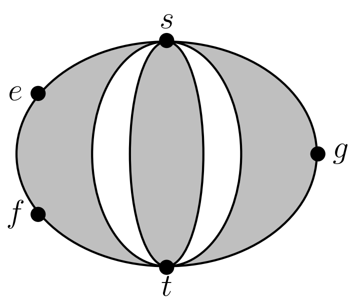


Figure 9.19: Lemma 9.6, item 2.

   or

4. *e, f, g lie in different triconnected primes. Then there exist different s, t, u of G such that for all planar embeddings of G:*

   (a) *s, t, u lie on a common boundary,*

   (b) *e, s, t lie on a common boundary,*

   (c) *f, t, u lie on a common boundary, and*

   (d) *g, s, u lie on a common boundary.*

   (e) *each pair of s, t, u lie on two common boundaries.*

Observe that in the last theorem whenever the phrase "lie on two common boundaries" occurs it may be replaced by "is a split pair in G'".

Proof. We assume that G is biconnected but not triconnected.

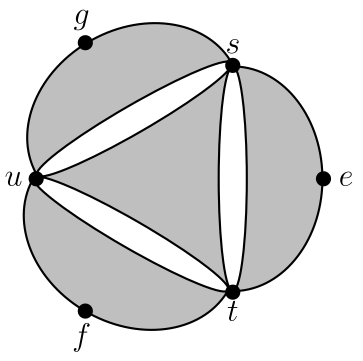


Figure 9.20: Lemma 9.6, item 3

1. (a) Since C is triconnected there is a unique planar embedding of C. Since the decomposition into triconnected primes is unique, G has a planar embedding such that e, f, g lie on a common boundary.

   (b) We assume that s, t belong to $G - X$. Since s, t lie on at least two common boundaries b_1, b_2, s, t are a split pair of G. Since e, f, s, t lie on a common boundary, w.l.o.g. we may assume that the common boundary is equal to b_1. Since s, g, t lie on a common boundary, w.l.o.g. we may assume that the common boundary is equal to b_2. Since s, t lie on b_1 and b_2 there are two disjoint paths P_1 and P_2, where P_1 is the path from s over e and f to t in b_1 and P_2 from s over g to t in b_2. Therefore, since s, t are split vertices of G there exists a planar embedding of G with a boundary b that consist of P_1 and P_2. Hence, there is a planar embedding of G with s, t, e, f, g on a common boundary. Observe that if s, t are terminals, then e, f, g are contained in a triconnected component. Furthermore, if one of s, t, say s, is a terminal $x \in X$, then in the paths P_1, P_2 the corresponding subpath s to x is contracted. Therefore, the same statement holds in this case.

   (c) We assume that s, t, u belong to $G - X$. Since s, t resp. s, u resp. t, u lie at least on two common boundaries b_1, b_2, resp. b_3, b_4 resp. b_5, b_6, s, t resp. s, u resp. t, u are split pairs of G. Furthermore, s, t, e resp. t, u, f resp. s, u, g lie on at least one common boundary b_7 resp. b_8 resp. b_9. W.l.o.g. we may assume that $b_1 = b_7$, $b_3 = b_8$ and $b_5 = b_9$.

   Therefore, w.l.o.g. we may assume that we have three disjoint paths P_1, P_2, P_3 where P_1 is the path from s over e to t in b_7, and P_2 is the path from t over f to u in b_8, and P_3 is the path from t over f to u in b_9.

   Since s, t, u lie on at least one common boundary, w.l.o.g. we may assume that there is a planar embedding with a boundary that consists of P_1, P_2, P_3. Hence, G has a planar embedding such that s, t, u, e, f, g lie on a common boundary.

   Observe that if s, t, u are terminals, then e, f, g are contained in a triconnected component. Furthermore, if two of s, t, u, say s, t, are different terminals $x, y \in$

X, then in the paths P_1, P_2, P_3 the corresponding subpaths between s and x resp. y (analogue for t and u) are contracted. The same holds if one of s, t, u is a terminal. Therefore, the statement also holds in this case.

2. Let Γ be the planar embedding with e, f, g on a common boundary. Since G is biconnected but not triconnected, we have the following cases.

   (a) e, f, g lie in a common triconnected component C. Then C has a unique planar embedding.

   (b) Two of e, f, g lie in a common triconnected component C_1, say e, f. Hence g lies in a different triconnected component C_2. Therefore, there exist a split pair s, t in G such that G splits into connected components by removal of s, t and C_1, C_2 lie in different connected components.

   Assume that w.l.o.g. $s = e$ and $t = f$. Then we have e, f, g in C_2, which contradicts the assumption.

   Therefore, at most one of s, t is a terminal. We assume w.l.o.g that s, t are no terminals. Since s, t are a split pair they lie on at least two common boundaries. Therefore, e, f, g, s, t lie on a common boundary. Since e, f belong to C_1, there is a boundary in C_1 that contains a path from s over e, f to t. Additionally, there is a boundary in C_2 that contains a path from s over g to t. Hence, e, f, s, t lie on a common boundary and g, s, t lie on a common boundary. Observe that if one of s, t is a terminal the same argument holds since we contract subpaths between $x \in \{s,t\}$ and $y \in X$ on the boundaries if $x = y$ and all other vertices of the subpath belong to $G - X \cup \{s,t\}$.

   (c) e, f, g belong to different triconnected components. We assume that no pair of e, f, g is a split pair. Therefore, there exists at least one split pair s, t such that e, f, g lie in different connected components by removal.

   If there is exactly one split pair with this property, then s, t lie on at least two common boundaries. Furthermore, s, e, t resp. s, f, t resp. s, g, t lie on a common boundary. Therefore, we have three disjoint paths P_1, P_2, P_3 such that P_1 is a path from s over e to t, P_2 is a path from s over e to t and P_3 is a path from s over g to t. This implies a $K_{2,3}[\{e,f,g\}]$-minor. By Theorem 9.10 there is no planar embedding with e, f, g on a common boundary, which leads to a contradiction.

   We assume that two of e, f, g lie in a common triconnected component C_1, say e, f, and g in an other triconnected component C_2. Since C_1 is triconnected, and e, f, g lie on a common boundary, there is a unique boundary in C_1 that contains e, f. Furthermore, this boundary has to contain s and t. Since e, f, g are contained in a common boundary and C_2 is triconnected, there is a unique boundary in C_2 that contains g, s, t. Since s, t is a split pair they lie on two common boundaries in G.

We assume that we have three split pairs. Therefore, we have three vertices s, t, u that connect each two of the three triconnected components of e, f, g. Obviously, s, t, u lie on at least one common boundaries. Therefore, s, t, u, e, f, g lie on a common boundary. W.l.o.g. we assume that s, t resp. t, u resp. s, u are the split pair of the triconnected component that contains e resp. f resp. g. Hence s, t, e resp. t, u, f resp. s, u, g lie on a common boundary. Furthermore, any pair of s, t, u lie on two common boundaries.

$\square$

A characterization for a $K_{2,3}[X]$-free graphs that is connected but not biconnected remains open. Obviously, a characterization for a $K_{2,3}[X]$-free graph that is not connected reduces to the characterization for the connected case: we consider each connected component independently.

We have the following theorem.

Theorem 9.11. *Given a connected planar graph G and three terminal vertices a, b and c. Let $X = \{a,b,c\}$. Let $t \in X$ and B a block of G: for t not in B, let v_t^B be the cut vertex that disconnects B from t by removal. If $a \in B$ (resp., $b \in B$, $c \in B$) we set $e^B = a$ (resp., $f^B = b$, $g^B = c$) otherwise $e^B = v_a^B$ (resp. $f^B = v_b^B$, $g^B = v_c^B$). Let $Y^B = \{e^B,f^B,g^B\}$.*

Then G does not contain a $K_{2,3}[X]$-minor if and only if

1. *G has no $K_{1,3}[X]$-minor, or*
2. *for each block B of G: $|Y^B| \leq 2$, or*
3. *for block B with $|Y^B| = 3$: B has no $K_{2,3}[Y^B]$-minor.*

Proof. We assume G to be connected but not biconnected. Otherwise Theorem 9.10 holds. Furthermore, since $K_{2,3}$ is biconnected, G has no $K_{2,3}$-minor if and only if each block of G has no $K_{2,3}$-minor. Additionally, for $t \in X$ and a block B let P_t^B be the union of all paths between t and v_t if $v_t \neq t$ otherwise $P_t^B = \emptyset$. Observe that if $P_t^B \neq \emptyset$ then all vertices of B has to use a path of P_t^B after traversing v_t^B to reach t since G is connected but not biconnected and v_t^B is a cut vertex of G. Therefore, B has a $K_{2,3}[X]$-minor after contracting all paths of $P_t^B \neq \emptyset$ if and only if B has a $K_{2,3}[Y^B]$-minor before contraction.

Therefore, G has no $K_{2,3}[X]$-minor if and only if each block B of G has no $K_{2,3}[Y^B]$-minor. Notice that $|Y^B| \leq 3$ for a block B of G since $|X| = 3$ for G.

1. (a) Since $K_{1,3}[X]$ is a proper subgraph of $K_{2,3}[X]$ G has no $K_{2,3}[X]$-minor.

   (b) For each block B of G: $|Y^B| \leq 2$. Since $|Y^B| \leq 2$, either at least one path of P_a^B, P_b^B or P_c^B contracts onto a terminal $t \in X$ that is a cut vertex in B or two of those paths contract onto a cut vertex v of B. Therefore, in the first case the vertices in B can reach at least one terminal of $X - t$ only using terminal t. Hence B induces no $K_{2,3}[Y^B]$-minor. In the second case, the vertices of B can reach at least two terminals not in B only using v. Hence B induces no $K_{2,3}[Y^B]$-minor. Therefore, for each block B we have no $K_{2,3}[Y^B]$-minor. Hence G has no $K_{2,3}[X]$-minor.

(c) For block B with $|Y^B| = 3$: B has no $K_{2,3}[Y^B]$-minor.

Assume that we contract each such block into a single vertex. Then for each block B of G: $|Y^B| \leq 2$ and therefore G has no $K_{2,3}[X]$-minor.

Therefore, since for each block B with $|Y^B| = 3$ B has no $K_{2,3}[Y^B]$-minor, G has no $K_{2,3}[X]$-minor.

2. G does not contain a $K_{2,3}[X]$-minor. Therefore, for each block B in G, B does not contain a $K_{2,3}[Y^B]$-minor. Therefore, either we have at most two terminals in B or the three terminals in B does not induce a $K_{2,3}[Y^B]$-minor. Therefore, we have either $|Y^B| \leq 2$ or for all blocks B with $|Y^B| = 3$, B contains no $K_{2,3}[Y^B]$-minor.

□

Observe that it exists at most one block with $|Y| = 3$ in a connected but not biconnected graph.

In Figures 9.21 and 9.22 two examples are given: the first example graph has a block B for that $|\{v_a,v_b,v_c\}| = 3$. B has a $K_{2,3}[v_a,v_b,v_c]$-minor that implies a $K_{2,3}[X]$-minor. The second example graph has exactly one block for that $|\{a,v_b,v_c\}| = 2$. Hence, it has no $K_{2,3}[X]$-minor.

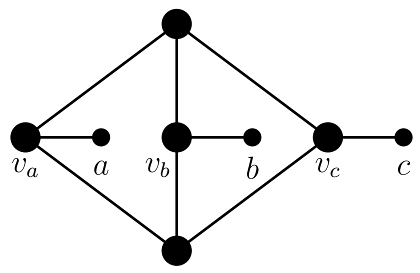


Figure 9.21: An example graph G with exactly one block B. B has a $K_{2,3}[v_a,v_b,v_c]$-minor that implies a $K_{2,3}[X]$-minor for G with $X = \{a,b,c\}$

Theorems 9.10 and 9.11 provide a characterization of $K_{2,3}[X]$-free planar graphs. We seek for an edge removal strategy such that a given graph fulfills the conditions of Theorems 9.10 and 9.11. An algorithm that solves all types of the characterization at minimum cost and takes the cheapest among them would give us the result. In Section 9.2.2 we investigated how to get the cheapest solution among the types I – IV. The algorithm runs in $O(n^3)$ time.

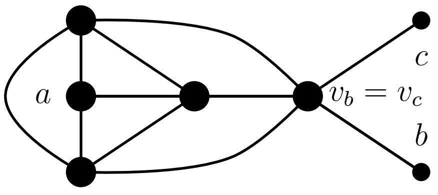
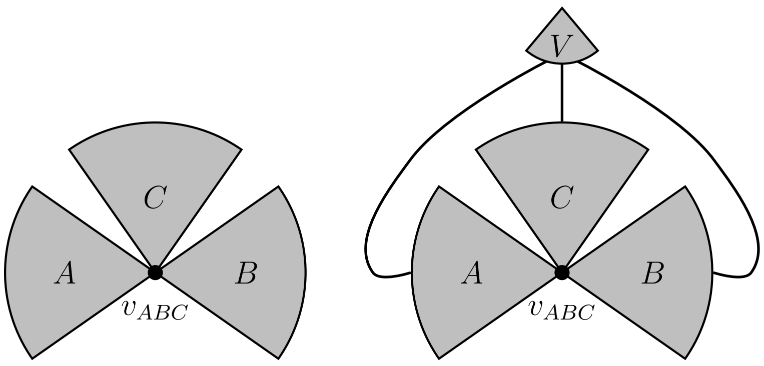


Figure 9.22: An example graph G with exactly one block B for that $|\{a,v_b,v_c\}| = 2$: B has no $K_{2,3}[a,v_b,v_c]$-minor that implies that G has no $K_{2,3}[X]$-minor with $X = \{a,b,c\}$

Let us analyze types presented in Theorems 9.10 and 9.11. First we begin with triconnected planar graphs. Obviously, a triconnected planar graph has a unique dual graph since it has a unique planar embedding Γ. By Theorem 9.10 first item, Γ has a face f such that a, b, c lie on its boundary. Therefore, the planar embedding Γ_D of the dual graph has a unique vertex v_{ABC} corresponding to f. Hence, we have the constellation visualized in Figure 9.23 that we name type V.



(a) Face A, B and C have at least one common vertex v_{ABC}

(b) Face V can reach A, B and C without crossing one of the terminal faces.

Figure 9.23: Illustration of case V.

Given a planar triconnected graph G we construct its extended dual graph such that we add a k-star for each of the faces A, B and C of the dual of G. We name the vertex of the k-star of A (B resp. C) $r(A)$ ($r(B)$ resp. $r(C)$). We set $\ell(e) = 0$ for all edges e of the k-stars. Our task is to determine a Steiner tree with the terminals $r(A)$, $r(B)$ and $r(C)$. When we contract the edges contained in the resulting Steiner tree we get a vertex v_{ABC} belonging to the boundaries of A, B and C. Transforming the edge contraction into the primal graph leads us to the deletion of the minimum number of edges in order to construct a planar subgraph of G that has a planar embedding with a, b and c on a common boundary.

Using the same techniques as in Section 9.2.2 we can prove the following theorem. See Figure 9.24.

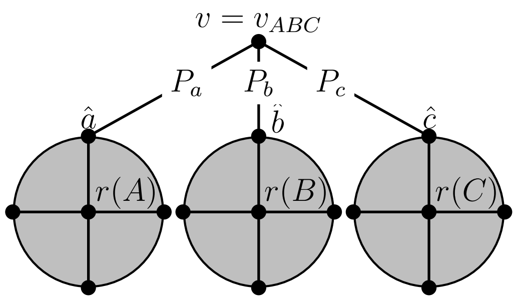


Figure 9.24: Solution of type V.

Theorem 9.12. *Assume that an optimal solution of problem K23-DEL is of type V. Let $T = P_a \cup P_b \cup P_c$ be the corresponding optimal set of edges and let x be the degenerated degree 3 vertex. Moreover, let*

$$\ell(P_f) \leq \ell(P_g) \leq \ell(P_h)$$

for $\{f, g, h\} = \{a, b, c\}$. Then the following holds:

- *$(P_f, r(F))$ is a shortest path from x to $r(F)$ in $G_D \circledast F$.*
- *$(P_g, r(G))$ is a shortest path from x to $r(G)$ in $G_D \circledast G/P_f$.*
- *$(P_h, r(H))$ is a shortest path from x to $r(H)$ in $G_D \circledast H/(P_f, P_g)$.*

Therefore, we determine the Steiner tree with terminals $r(A)$, $r(B)$, $r(C)$ in the following way. See Figure 9.24:

For a vertex v in G_D we contract the shortest path from v to $r(A)$. Since edges of the k-star of A have weight 0, this is equal to determining the shortest path from v to the closest vertex $\hat{a}$ on the boundary of A, say P_a. In an analogous way we calculate the corresponding shortest path P_b resp. P_c from v to the closest vertex on the boundary of B resp C, say $\hat{b}$ resp. $\hat{c}$. We contract P_b and P_c after each other. Notice that P_b resp. P_c might differ from the corresponding shortest path in the original G_D. Since we contract P_a first, P_b corresponds to the shortest path from the vertices of P_a to $\hat{b}$. Finally, P_c corresponds to the shortest path from the vertices of P_a and P_b to $\hat{c}$. Obviously, the resulting contraction is a feasible solution of the type V resp. VI, but must not be the cheapest one. We have to consider every ordering of $\{a, b, c\}$ for the shortest path calculation. We do this for every vertex of G_D to get the cheapest solution.

See Algorithm 32 for a pseudo-code description.

We bound the complexity of Algorithm 32: The degree 3 vertex $v = v_{ABC}$ can be determined in $O(n)$ time since we have to visit each vertex of G_D once. Additionally, there are $3! = 6$

Algorithm 32: Determination of an optimal solution of type V: Output is an optimal solution of type V.

```
forall the vertex $v \in V$ do
    forall the orderings $(f, g, h)$ with $\{f, g, h\} = \{a, b, c\}$ do
        Determine shortest path $P_f$ from $v$ to $r(F)$;
        Contract $P_f$;
        Determine shortest paths $P_g$ from $v$ to $r(G)$;
        Contract $P_g$;
        Determine shortest paths $P_h$ from $v$ to $r(H)$;
        $\mathrm{Sol}(f, g, h) \Leftarrow P_f \cup P_g \cup P_h$;
        $\mathrm{val}(f, g, h) \Leftarrow \ell(\mathrm{Sol}(f, g, h))$;
    $\mathrm{val}(x, y) \Leftarrow \min\{\mathrm{val}(f, g, h) \mid \{f, g, h\} = \{a, b, c\}$ and $\mathrm{Sol}(f, g, h)$ is feasible$\} =$
    $\mathrm{val}(f^*, g^*, h^*)$;
    $\mathrm{Sol}(x) \Leftarrow \mathrm{Sol}(f^*, g^*, h^*)$;
$\mathrm{val} \Leftarrow \min\{\mathrm{val}(x) \mid x \in V\} = \mathrm{val}(x^*)$;
Return: $\mathrm{Sol} \Leftarrow \mathrm{Sol}(x^*)$;
```

possible orderings of the three terminals. For each ordering we have to solve three shortest path problems that can be done in $O(n)$ time.

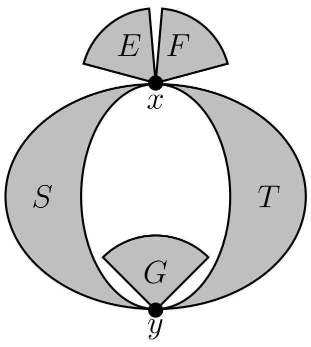
Therefore, the Algorithm 32 has a $O(n^2)$ running time.

Comparing the optimal solution E'_{IV} of type IV with the optimal solution E'_V of type V resp. E'_{VI} of type VI we get $\ell(E'_V) \leq \ell(E'_{IV})$ resp. $\ell(E'_{VI}) \leq \ell(E'_{IV})$ since one of the Steiner trees of type IV induces already a feasible solution of type V resp. VI. Therefore, the optimal solution of type IV is dominated by the optimal solutions of types V and VI.

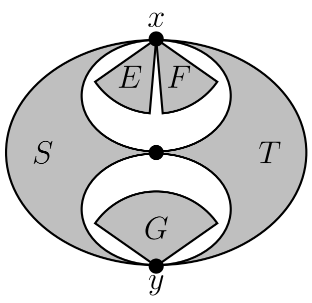
The next constellation type VI is introduced by Theorem 9.10 item 2: s, t is a split pair such that e, f lie in one triconnected component and g lie in an other triconnected component. Furthermore, e, f, s, t are on a common boundary in every planar embedding of G. The same holds for s, t, g. Obviously, a solution is that e, f, g have a common boundary for a given planar embedding. See constellation of type V. Notice that this solution might not be optimal. Therefore, we have additionally the cases visualized in Figure 9.25.

Hence, we have to solve the following minimization problem: Given a planar embedding of a biconnected planar graph G we construct its extended dual graph D such that we add a k-star for each of the faces E, F, G of the dual of G. We name the vertex of the k-star of E (F resp. G) $r(E)$ ($r(F)$ resp. $r(G)$). We set $\ell(e) = 0$ for all edges e of the k-stars. Additionally, we have two different faces S, T of D. In an analogous way we add k-stars and get vertices $r(S)$, $r(T)$. Our task is to determine a Steiner forest with the terminals $r(E)$, $r(F)$, $r(G)$, $r(S)$, $r(T)$ and the nets $\mathcal{N}_1 = \{r(E), r(F), r(S), r(T)\}$, $\mathcal{N}_2 = \{r(G), r(S), r(T)\}$ and $\mathcal{N}_3 = \{r(S), r(T)\}$ such that S, T separates E, F from G. Notice that the trees of the Steiner forest do not need to be disjoint.

Since the number of terminals and nets is bounded we conjecture that the optimal solution



(a) Faces S, T, E, F resp. S, T, G have a common vertex x resp. y and S, T separates E, F from G



(b) Faces S, T, E, F resp. S, T, G have a common vertex x resp. y and S, T that have an additional common vertex separates E, F from G

Figure 9.25: Illustration of case VI.

of type VI can be solved in polynomial time.

In an analogous way we solve the edge-deletion problem introduced by Theorem 9.10 item 3 that we name constellation type VII; see Figures 9.26 to 9.28:

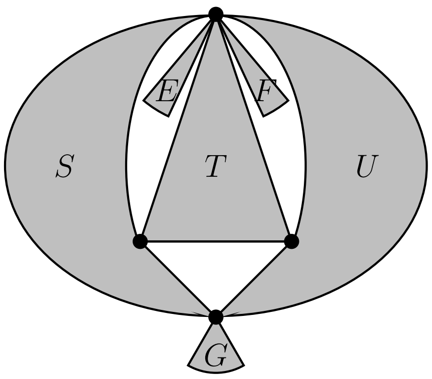


Figure 9.26: Lemma 9.6, item 3: constellation of type VII a

Notice that the types VII a - c visualized in the Figures 9.26 - 9.28 are equal under 2-isomorphism. Again, one solution is type V but do not need to be an optimal one. Additionally, we have three instead two as in the previous case different faces S, T, U of D. After the addition of the corresponding k-stars we get $r(S)$, $r(T)$, $r(U)$.

Our task is to determine a Steiner forest with the terminals $r(E)$, $r(F)$, $r(G)$, $r(S)$, $r(T)$, $r(U)$ and the nets

- $\mathcal{N}_1 = \{r(E),r(S),r(T)\}$,

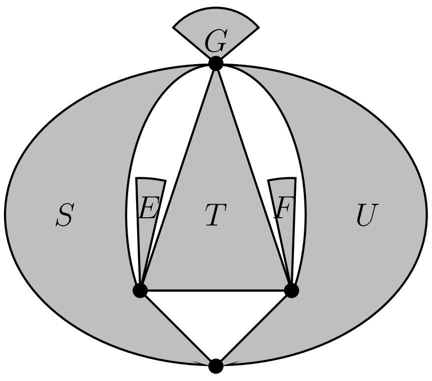


Figure 9.27: Lemma 9.6, item 3: constellation of type VII b

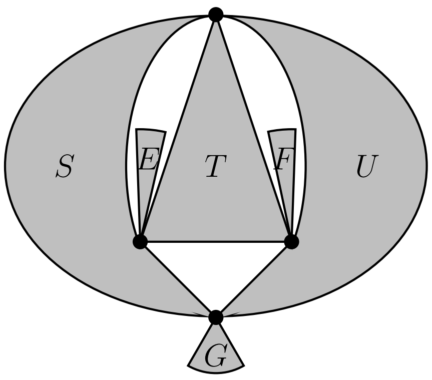


Figure 9.28: Lemma 9.6, item 3: constellation of type VII c

- $\mathcal{N}_2 = \{r(F), r(T), r(U)\}$,
- $\mathcal{N}_3 = \{r(G), r(S), r(U)\}$,
- $\mathcal{N}_4 = \{r(S), r(T), r(U)\}$,

such that

- S, T separates E from F, G,
- S, U separates G from E, F, and
- T, U separates F from E, G.

Notice that the trees of the Steiner forest do not need to be disjoint.

Since the number of terminals and nets is bounded we conjecture that the optimal solution of type VII can be solved in polynomial time.

Unfortunately, the characterization for rooted $K_{2,3}$-minor free planar graphs has not lead yet to a polynomial algorithms for the corresponding weighted minimum deletion problem, only some sub-cases could be solved in polynomial time. What makes this case far more difficult is the fact that it is even unclear whether the number of types for characterizing *all* planar embeddings in a rooted $K_{2,3}$-minor free planar graph could be bounded polynomially.

Chapter 10

Bimodal Crossing Minimization

The importance of automatic graph drawing stems from the fact that many different types of data can be modeled by graphs. In most applications, the interpretation of an edge is asymmetric, so that the graph is intrinsically directed. This is the case, e.g., for metabolic networks. Here, the incoming edges of a reaction vertex correspond to reactants, while the outgoing edges correspond to products of the modeled reaction. Consequently, a good layout of such a network should separate incoming from outgoing edges, e.g., by letting the incoming edges enter on one side of the vertex and letting the outgoing edges leave on the opposite side. By this, the human viewer is able to distinguish reactants from products much more easily; see Figure 10.1.

In spite of its practical relevance, the direction of edges is ignored by many graph drawing algorithms. The graph is processed as an undirected graph first; only after the positions of vertices and edges have been determined the direction is visualized by replacing lines by arrows. An important exception is given by hierarchical drawings, in which incoming and outgoing edges are separated by definition. Furthermore, a polynomial time algorithm for hierarchical drawings of digraphs that allows directed cycles and produces the minimum

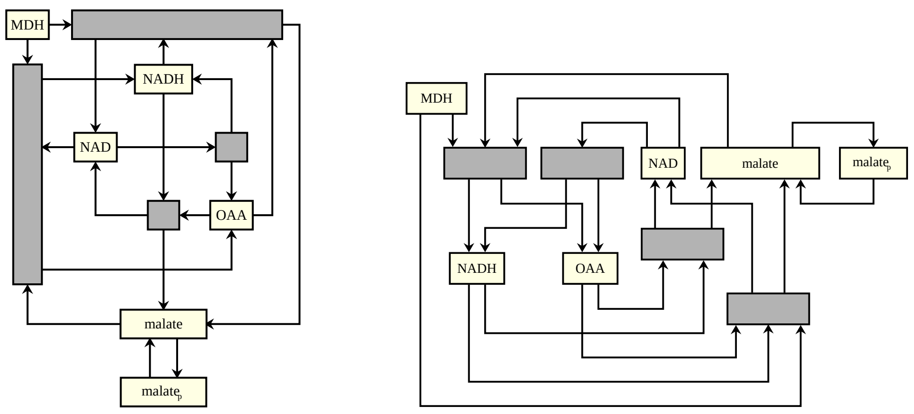


Figure 10.1: Two drawings of the same graph, both have three crossings, with unsorted (left) and sorted (right) incoming and outgoing edges; gray vertices represent reactions

number of bends is given by Bertolazzi, Di Battista and Didimo in [5]. However, the restriction to this special type of drawing may lead to many more crossings than necessary.

In this chapter, our aim is to adapt the planarization method in order to obtain the desired separation of incoming and outgoing edges.

This is joint work with Christoph Buchheim, Michael Jünger and Annette Menze, published as a conference paper [10]. The implementation of the used heuristics is by Maria Kandyba.

We focus on the planarization step, i.e., the computation of a planar embedding of the graph after possibly adding artificial vertices representing edge crossings. The objective is to add as few such vertices as possible. For a comprehensive survey over the planarization approach, see [63].

In order to obtain the separation of edges, we consider the additional bimodal restriction that all incoming edges appear consecutively in all cyclic adjacency lists. We show how to adapt the well-known approach based on finding a planar subgraph first and then reinserting the missing edges one after the other in a very efficient way. We use an experimental evaluation to investigate the question of how many additional crossings have to be expected from restricting the class of feasible embeddings in this way. The results show that—for practical instances—this increase is usually negligible.

We do not address the question of how to realize the resulting embedding by an actual drawing of the graph. Notice however that once we have such an embedding at hand, it is easily possible to adapt, e.g., the orthogonal layout algorithm such that incoming and outgoing edges lie on opposite sides [67].

In Section 10.1 we recall the concept of bimodality and describe the basic transformation used by the evaluated algorithms. Next we propose a postprocessing technique that can be combined with any crossing reduction approach, see Section 10.2. Then we look into the planarization method; the problem of finding a planar subgraph is considered in Section 10.3, while edge reinsertion is dealt with in Section 10.4. In Section 10.5, we discuss a recently developed exact approach for crossing minimization. In Section 10.6, we present an experimental evaluation showing that the number of crossings computed by different methods does not grow much due to our additional requirement.

10.1 Bimodal Embeddings

An embedding of a graph $G = (V, E)$ is called *bimodal* if and only if for every vertex v of G the circular list of the edges around v is partitioned into two (possibly empty) linear lists of edges, one consisting of the incoming edges and the other consisting if the outgoing edges. A planar digraph is *bimodally planar* if and only if it has a bimodal embedding that is planar. This structure was first investigated by Bertolazzi, Di Battista, and Didimo in [5]. Bimodal planarity of a graph G can be decided by testing planarity of a simple transformation of G in $O(|V|)$ time [5]. The transformation is applied in the following way: for every vertex v of G expand v by an expansion edge e and add all incoming edges of

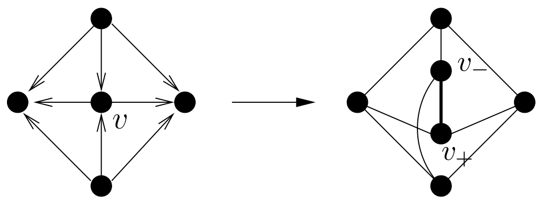


Figure 10.2: A directed graph G and its d-graph G_d. The bold edge is an expansion edge. Notice that G_d is equal to $K_{3,3}$

v to one end vertex v_- of e and all outgoing edges to the other end vertex v_+ of e. The resulting graph is denoted by $G_d = (V_d, E_d)$ in the following. We call G_d the *d-graph* of G. An illustration of this construction is given in Figure 10.2.

Throughout this chapter, we will denote the set of all expansion edges by E'. We use this simple transformation for adapting techniques for undirected crossing minimization to the directed variant. Planar directed graphs are not necessarily bimodally planar. By Kuratowski's theorem, this can only happen if a $K_{3,3}$ or K_5 subdivision is created by the transformation into a d-graph. For graphs with all vertices of degree at most three the transformation of G to G_d is trivial, as no vertices are split in this case. In particular, this holds for cubic graphs that are defined by the property that all vertices have degree three. Therefore, a directed cubic graph is bimodally planar if and only if it is planar. This is also true for graphs in which each vertex has at most one incoming edge or at most one outgoing edge.

10.2 Naive Post-Processing Approach

We first discuss a post-processing procedure that can be used after applying any crossing reduction algorithm or heuristic to the d-graph G_d. Our aim is to embed G_d such that no expansion edge crosses any other edge; contracting all expansion edges then yields an embedding of G with the desired separation of incoming and outgoing edges.

So assume that any embedding of G_d is given. We first delete all edges crossing any expansion edge. If two expansion edges cross each other, we delete one of them. Next, we reinsert all deleted edges one after another, starting with the deleted expansion edges. As explained in Section 10.4.1 below, we can insert a single edge with a minimum number of crossings for the fixed embedding computed so far such that crossings with expansion edges are prevented. If reinserting an expansion edge produces any crossings, the crossed (non-expansion) edges have to be deleted and put to the end of the queue of edges to be reinserted.

At the end of this reinsertion process, no expansion edge will cross any other edge. However, the number of crossings of the remaining edges might grow significantly in this approach. In the following sections, we explain how to get better results by adapting well-known crossing minimization approaches for our purposes, especially the planarization approach.

10.3 Maximum Bimodally Planar Subgraphs

Recall from Chapter 9 that the maximum planar subgraph problem—the problem of finding a planar subgraph of a given graph that contains a maximum number of edges—is $\mathcal{NP}$-hard [64, 86, 85]. Recently, it was shown that this remains true even for cubic graphs:

Theorem 10.1 (Faria et al. [28]). *The maximum planar subgraph problem is $\mathcal{NP}$-hard for cubic graphs.*

In Chapter 9 we observed that this is even true for K_5-minor or $K_{3,3}$-minor, respectively, free graphs.

As a cubic graph is equal to its d-graph, we derive that this also holds for the maximum bimodally planar subgraph problem:

Corollary 10.1. *It is an $\mathcal{NP}$-hard problem to compute a maximum bimodally planar subgraph of a directed graph, even for a cubic graph.*

For computing *maximal* bimodally planar subgraphs, i.e., bimodally planar subgraphs such that adding any further edge of G destroys bimodally planarity, we do the following: it is easy to see that the bimodally planar subgraphs of G are in one-to-one correspondence to the planar subgraphs of G_d containing all expansion edges. Thus we have to modify a given maximal planar subgraph algorithm such that it never deletes any expansion edge. Methods for finding maximal planar subgraphs have been studied intensively [23, 50, 59]; here we only discuss the incremental method; see Section 10.3.1. We also have a look at the exact approach; see Section 10.3.2.

10.3.1 Incremental Method

Starting with the empty subgraph $(V_H, \emptyset)$ of some graph $H = (V_H, E_H)$, the incremental method tries to add one edge from E_H after the other. Whenever adding an edge would destroy planarity, it is discarded, otherwise it is added permanently to the subgraph being constructed. The result is a maximal planar subgraph of H, which however is not a maximum planar subgraph in general.

To find a maximal bimodally planar subgraph of G, we have to compute a maximal planar subgraph of its d-graph G_d. However, this subgraph must always contain all expansion edges, so that the latter can be contracted at the end. We thus have to start with the subgraph (V_d, E')—which is obviously planar—instead of the empty subgraph $(V_d, \emptyset)$. Then we try to add the remaining edges $E_d \setminus E'$ as before. The resulting subgraph of G_d corresponds to a maximal bimodally planar subgraph H of G.

10.3.2 Exact Method

An exact approach for finding a maximum planar subgraph of $H = (V_H, E_H)$ based on polyhedral techniques was devised in [55]. The problem is modeled by an integer linear program (ILP) as follows: for every edge $e \in E_H$, a binary variable x_e is introduced, having value one if and only if e belongs to the chosen subgraph. To enforce that the modeled subgraph is planar, one has to make sure that it contains no Kuratowski subgraph of H, i.e., no subdivision of K_5 or $K_{3,3}$. In terms of the model, this is equivalent to the constraint

$$\sum_{e \in K} x_e \leq |K| - 1$$

for every (edge set of a) Kuratowski graph K in G. As we search for a planar subgraph containing the maximal number of edges, the number of variables set to one should be maximized. The integer linear program is thus

$$\begin{array}{llll} \max & \sum_{e \in E_H} x_e & & \\ \text{s.t.} & \sum_{e \in K} x_e \leq & |K| - 1 & \text{for all Kuratowski subgraphs } K \text{ of } G \\ & x_e \in & \{0, 1\} & \text{for all } e \in E_H \ . \end{array} \tag{10.1}$$

This integer linear program can now be solved by branch-and-cut. However, in order to improve the runtime of such algorithms and hence obtain a practical solution method, one further has to investigate this formulation and exhibit other classes of valid inequalities as well as fast techniques for finding violated constraints for a given fractional solution. For details, the reader is referred to [55].

This solution approach can easily be adapted to our new situation: we only have to ensure that the edges in E' always belong to the chosen subgraph. In other words, we only have to add the constraint $x_e = 1$ to the ILP (10.1) for each edge $e \in E'$. Observe that this type of constraint is harmless with respect to the complexity of the problem, as it cuts out a face from the polytope spanned by the feasible solutions of (10.1).

10.4 Edge Reinsertion

After calculating a maximal (resp., maximum) bimodally planar subgraph, the deleted edges have to be reinserted. Our objective is to reinsert them one by one so that the minimum number of crossings are produced for each edge.

This can be done in two different ways: either by inserting an edge into a fixed bimodally planar embedding of the bimodally planar subgraph, see Section 10.4.1, or by inserting an edge optimally over all bimodally planar embeddings of the bimodally planar subgraph, see Section 10.4.2. Again, we have to treat expansion edges differently, as they may not be involved in any edge crossings.

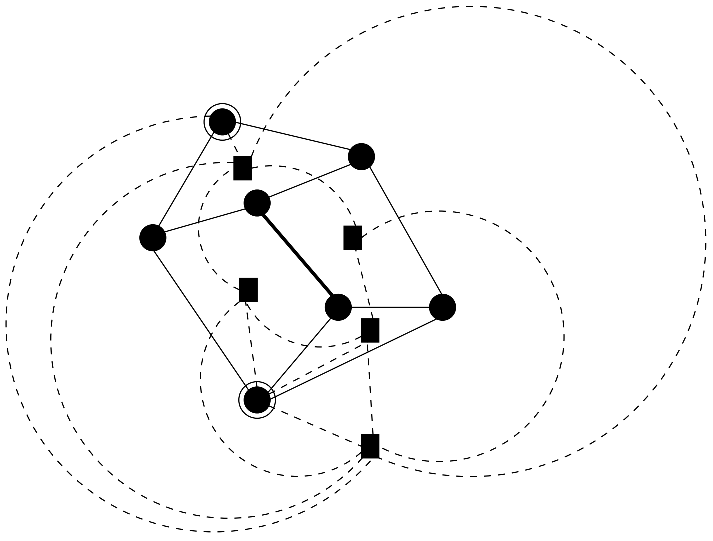


Figure 10.3: A maximal planar subgraph with its extended dual graph. Edge $e = (v, w)$ has to be reinserted (v and w are circled). The bold edge is an expansion edge and has no dual

10.4.1 Fixed Embedding

Given a fixed embedding $\Gamma(G_d)$ of G_d, it is easy to insert an edge $e = (v, w)$ into $\Gamma(G_d)$ such that a minimal number of crossings is produced. For this, one can use the *extended dual graph* D of $\Gamma(G_d)$, the vertices of which are the faces of $\Gamma(G_d)$ plus two vertices v_D and w_D corresponding to v and w. For each edge in $E_d \setminus E'$, we have the dual edge in D. Additionally, we connect v (resp., w) with all vertices in D corresponding to faces that are adjacent to v (resp., w) in $\Gamma(G_d)$.

Then we calculate the shortest path from v to w in the extended dual graph and insert the edge e into $\Gamma(G_d)$ along this path, replacing crossings by dummy vertices. Clearly, the shortest path does not cross any edge of E' as its dual edge is not included in D. This can be done in $O(|V|)$ time.

10.4.2 All Embeddings

In the previous section we have considered reinserting an edge into a fixed embedding. For getting fewer edge crossings, a powerful method is to calculate the shortest path between two vertices v and w over all embeddings. In [43] a linear time algorithm is presented for finding an optimal embedding which allows to insert e with the minimum number of crossings. It uses the SPQR-tree BC-tree data-structures for representing all planar embeddings of a connected graph.

In the same straightforward way as explained in the previous section, this approach can be adapted such that no expansion edge is crossed by any reinserted edge. The resulting algorithm runs in $O(|V|)$ time.

10.5 Exact Directed Crossing Minimization

It is a well-known fact that the general crossing minimization problem for undirected graphs is $\mathcal{NP}$-hard [34]. More recent results show that this is even true for graphs with all vertices of degree three:

Theorem 10.2 (Hliněný [48], Pelsmajer, Schaefer, Štefankovič [69])**.** *The crossing minimization problem is $\mathcal{NP}$-hard for cubic graphs.*

Corollary 10.2. *It is an $\mathcal{NP}$-hard problem to compute a drawing of G separating incoming and outgoing edges such that the number of crossings is minimal. This even holds for cubic graphs.*

Despite the $\mathcal{NP}$-hardness of undirected crossing minimization, an exact approach has been devised recently [9]; a branch-and-cut algorithm is proposed for minimizing the number of crossings over all possible drawings.

The first step in this approach is to replace every edge of the graph by a path of length (at most) $|E|$. After this, one may assume that every edge has a crossing with at most one other edge. The ILP model used in this approach contains a variable x_{ef} for all pairs of edges $(e, f) \in E \times E$, having value one if and only if there is a crossing between e and f in the drawing to be computed. By appropriate linear constraints, one can ensure that the given solution is realizable, i.e., corresponds to some drawing of G.

Again, it is easy to adjust this method to our problem, i.e., the problem of computing a crossing-minimal drawing with incoming and outgoing edges separated. For this, we can apply the above algorithm to the graph G_d. Then we only have to make sure that the expansion edges do not have any crossings in the computed solution. We can thus do the adjustment as follows: first observe that the edges in E' do not have to be replaced by a path at all, as they are not allowed to produce crossings. Now we can just omit the variable x_{ef} whenever $e \in E'$ or $f \in E'$, and thereby set this variable to zero implicitly. The resulting ILP will thus have exactly the same number of variables as the original ILP for the non-transformed graph. It will not become harder structurally, as it arises from setting variables to zero.

10.6 Experimental Comparison

In the previous sections, we showed how to adapt several crossing minimization algorithms and heuristics in a simple way such that for directed graphs the sets of incoming and outgoing edges are separated in the adjacency lists. This is obtained by transforming the original directed graph into a new undirected graph where certain edges do not allow any crossings. From the nature of this transformation and the described modifications, it is obvious that the runtime is not affected negatively. We also observed this in our experiments.

For this reason, we focused on the number of crossings in the evaluation reported in the following: we are interested in comparing the number of crossings when (a) the direction of edges is ignored, i.e., crossing minimization is done as usual, and (b) we apply the transformation in order to separate incoming from outgoing edges. Theoretically, the crossing number cannot decrease by our modification, but it is possible that it grows considerably.

However, our experiments show that for practical graphs the number of crossings is not increased significantly. In fact, the increase in the number of crossings is marginal compared with the variance due to the randomness of the heuristics, such that for many instances the number of crossings after the transformation even decreases. Combining this observation with the simpleness of implementation and the fact that runtime does not increase, our claim is that these techniques should always be applied when dealing with (meaningfully) directed edges.

For the experiments, we used the instances of the Rome library of directed graphs [74], consisting of two sets of graphs called `north` and `random`. The former contains 1277 directed acyclic graphs on 10 to 100 vertices derived from real-world instances. The latter contains 909 directed acyclic graphs randomly generated in a specific way, they are much denser in general.

We first applied the simple incremental method (Section 10.3.1) combined with the optimal edge reinsertion over all embeddings (Section 10.4.2). As mentioned above, it turned out that the increase in the number of crossings when separating incoming and outgoing edges is very small in general. This is shown in Figure 10.4 (a) and (b), where each instance is given by a plus sign. Its x-coordinate is the number of crossings before the transformation and the y-coordinate is the number of crossings afterwards. In particular, each cross on the diagonal line represents an instance with the same number of crossings before and afterwards. A cross above the diagonal represents an instance for which the number of crossings increases. Due to the randomness of the heuristics, there are also crosses below the diagonal, in particular for the `random` instances.

Another interesting finding is the negligible increase in the number of crossings for planar graphs: if G is planar, then G_d is not necessarily planar. Anyway, if we consider all 854 planar `north` instances, then the average number of crossings after the transformation is only 0.04, i.e., in most cases the graph remains planar. The set of `random` instances does not contain any planar graph.

We next applied the optimal planar subgraph method (Section 10.3.2), again in combination with the optimal edge reinsertion over all embeddings (Section 10.4.2). As many instances could not be solved within a reasonable running time, we had to set a time limit of five CPU minutes (on an Athlon processor with 2.0 GHz). Within this time limit, 89 % of the `north` instances and 33 % of the `random` instances could be solved. The results are

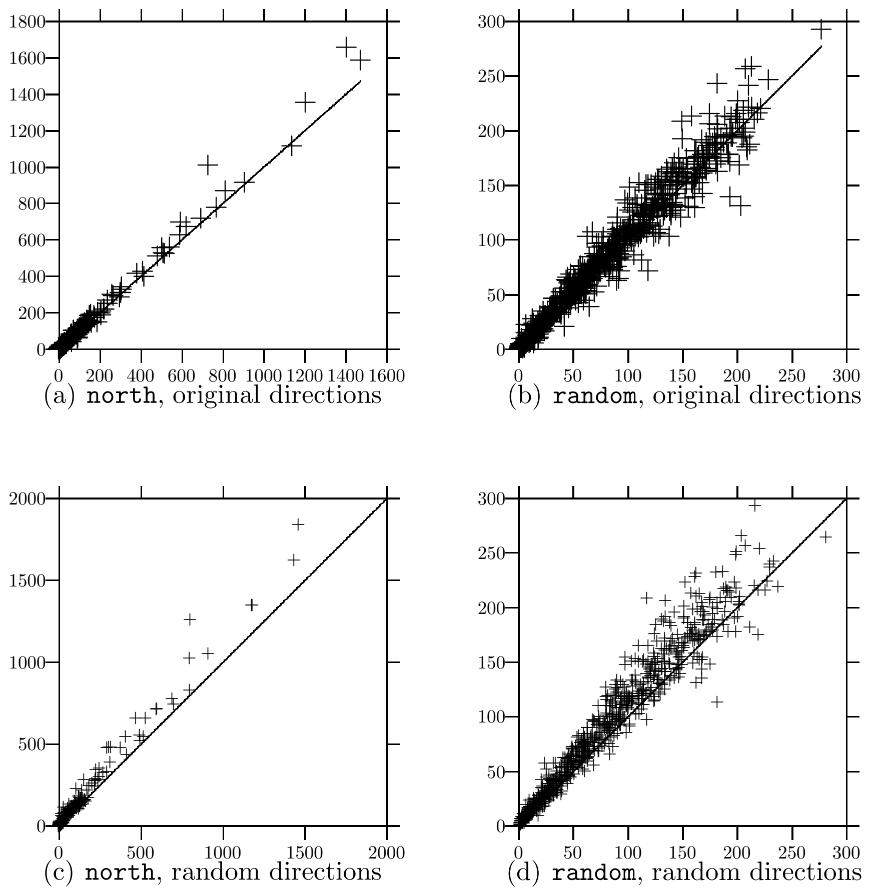
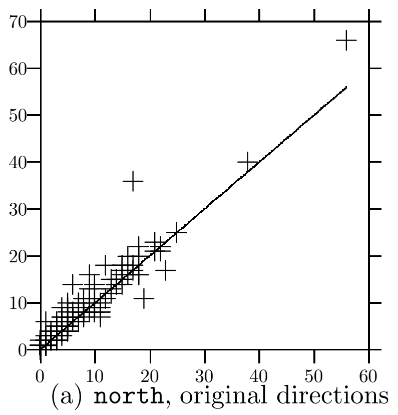


Figure 10.4: Numbers of crossings before and after the transformation, using the incremental planar subgraph heuristic. For non-planar graphs, the average increase is 0.36 % (a), 0.59 % (b), 0.95 % (c), and 0.87 % (d), respectively.

shown in Figure 10.5; the general picture is similar to the one for the incremental method.

The directed graphs contained in the libraries `north` and `random` are all acyclic. This fact might favor a small number of additional crossings. For this reason, we also examined graphs with a random direction for each edge. To allow us to compare the corresponding results to the results presented so far, we used the `north` and `random` instances again, this time with the direction of each edge reversed with a probability of 1/2. The results obtained with the incremental heuristic are displayed in Figure 10.4 (c) and (d). In fact, the increase in the number of crossings induced by sorting adjacency lists is more obvious now compared to Figure 10.4 (a) and (b), but it is still very small.

Nevertheless, we conjecture that in theory the requirement of separating incoming and outgoing edges may induce a quadratic number of edge crossings even for planar graphs. We have constructed a family of directed planar graphs G_k such that G_k has $O(k)$ edges and


(a) `north`, original directions

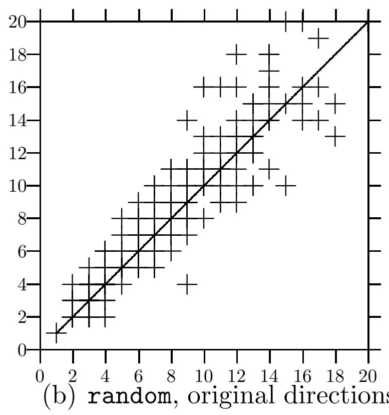

(b) `random`, original directions

Figure 10.5: Numbers of crossings before and after the transformation, using the optimal planar subgraph method. For non-planar graphs, the average increase is 3.30 % (a) and 4.21 % (b), respectively.

such that the planarization heuristic has always produced $\Omega(k^2)$ crossings when separating incoming from outgoing edges.

The graph G_k is defined as follows: it consists of two wheel graphs W_{2k} sharing their rim; one of them has all spokes directed from the rim to the hub, the other one has spokes with alternating direction.

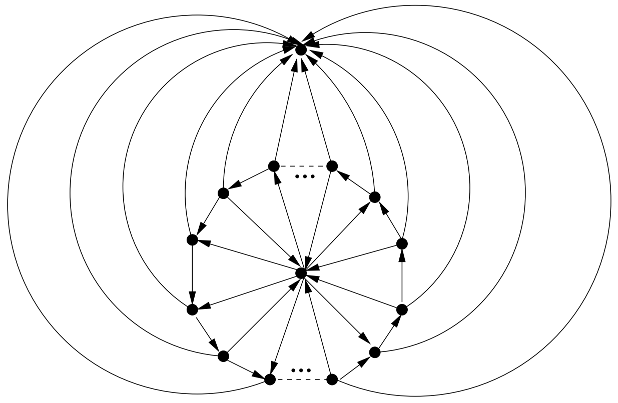


Figure 10.6: The graph G_k consisting of two wheel graphs W_{2k} sharing their rim

We applied the planarization method to the graphs G_k many times, with enforced separation of incoming and outgoing edges (an example for $k = 30$ is shown in Figure 10.7). For all k, the smallest number of crossings we could find was

$$\sum_{i=1}^{k} \lfloor i/2 \rfloor = \Theta(k^2).$$

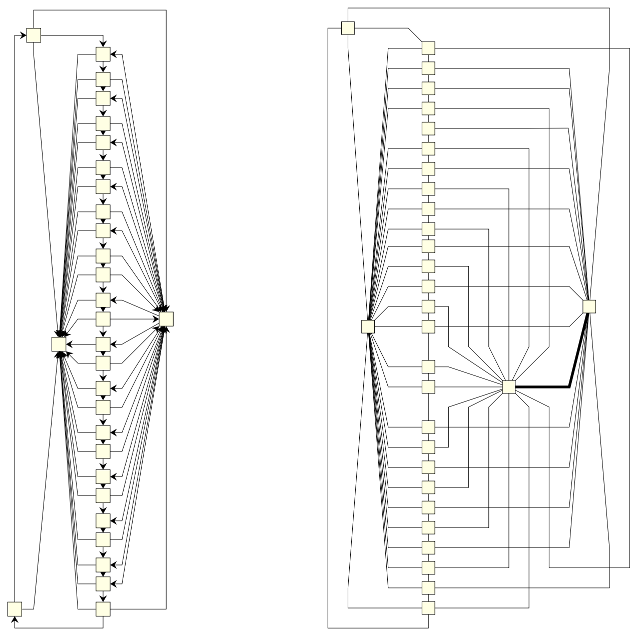


Figure 10.7: A planar drawing of G_{30} (left) and the bimodal drawing with the minimal number of crossings 42 obtained by applying the planarization heuristic on its d-graph, the expansion edge is visualized bold (right)

We conjecture that this is the minimum number of crossings for all bimodal drawings of G_k. This would mean that a quadratic number of crossings is unavoidable even for planar graphs.

Part III

Conclusion and Future Work

In this part we summarize the results presented in this work and give an overview on the remaining open problems.

The most interesting problem of the c-planarity research field is to state the computational complexity of deciding c-planarity for general clustered graphs. In Chapter 4 we discussed the polynomial relation between a specific topological inference problem and deciding c-planarity. In Chapter 7 we have given some non-trivial polynomially solvable cases. As a resulting observation, it seems to be difficult to extract subclasses that are in $\mathcal{P}$. Combining this observation with the characterizations made in Chapter 5 it would be surprising if we could decide in polynomial time whether a given planar clustered graph is c-planar even when restricted to simpler subcases such as

- G has a fixed embedding and T has only one layer (excluding the root cluster),
- G has a fixed embedding,
- G is series-parallel.

If G is a tree or outer-planar we conjecture that it is possible to decide c-planarity of a given planar clustered graph in polynomial time. For the second subcase, the reason is that an outerplanar graph has no minors isomorphic to K_4 and $K_{2,3}$ [81] that means that the corresponding SPQR-tree has only S-, P- and Q-nodes. Further, in an SPQR-tree each skeleton of a P-node has exactly three edges where one edge is an original one. Consequently, for any cluster ν in T we have always a path in each skeleton of a P-node that belongs to $G - G(\nu)$ and this path is equal to an original edge of G. Therefore, for a skeleton of an S-node the reference edge always represents a path in $G - G(\nu)$. Unfortunately, there is no restriction how the vertices of a cluster lie in the pertinent graph of the S-node. However, we expect the number of feasible edge-augmentations in order to get the clustered graph c-connected and planar to be low.

One good strategy might be to code the c-planarity problem in an ILP as a crossing minimization problem and to check experimentally how hard it is to solve. In the following we discuss and propose an idea for future work.

It is a well-known fact that the general crossing minimization problem for undirected graphs is $\mathcal{NP}$-hard [34]. Since cluster crossing minimization is a generalization we get the following corollary.

Corollary 10.3. *It is an $\mathcal{NP}$-hard problem to compute a drawing of a clustered graph such that the number of crossings is minimal. This even holds for underlying graphs that are cubic.*

Despite the $\mathcal{NP}$-hardness of crossing minimization, an exact approach has been devised recently [9]; a branch-and-cut algorithm is proposed for minimizing the number of crossings over all possible drawings.

The first step in this approach is to replace every edge of the graph by a path of length (at the most) $|E|$. After this, one may assume that every edge has a crossing with at most one other edge. The ILP model used in this approach contains a variable x_{ef} for all pairs of edges $(e, f) \in E \times E$, having value one if and only if there is a crossing between e and f in the drawing to be computed. By appropriate linear constraints, one can ensure that the given solution is realizable, i.e., corresponds to some drawing of G.

In this part we adjust this method to our problem, i.e., the problem of computing a crossing-minimal drawing of a clustered graph $C = (G, T)$. Before we start with exploiting the terminology of crossing minimization for clustered graphs we have to investigate the terminology of c-planarity under the aspects of crossings.

The next theorem gives us a characterization for the case that no crossings are allowed in the clustered graph C, say C is c-planar.

Theorem 10.3. *Let $C = (G, T)$ be a clustered graph. Let $R = \{R(\nu) \mid \nu \in T\}$ be a set of cycles defined as follows: for each cluster ν, there is a cycle $R(\nu)$ of length the number of incident edges of ν.*

C is c-planar if and only if there exists a drawing D of $G \cup R$ such that D has only the following crossings: for every cluster ν there is a bijective mapping between incident edges of ν and edges of $R(\nu)$ and each assignment corresponds to a crossing in D.

Proof. If C is c-planar then C has a c-planar embedding Γ. Consequently, G is planar and there are neither region-edge crossings nor region-region crossings. By definition of c-planarity, for every cluster ν its incident edges are ordered consecutively in Γ. Furthermore, they are crossing the boundary of a closed cluster region. Hence, the boundary can be modelled as a cycle that corresponds to $R(\nu)$. Therefore, there is a drawing D of $G \cup R$ such that each edge in $R(\nu)$ is crossed exactly once by an incident edge of ν. Further, an incident edge of ν is crossed exactly once by an edge of $R(\nu)$. Consequently, we have a bijective mapping of crossings between incident edges of ν and edges of $R(\nu)$ in D. Further, there are no additional crossings between an edge of G and an edge of a cycle in R.

We have given a drawing D of $G \cup R$ such that G and R are each plane in D and for each cluster ν there is a bijective mapping of crossings between its incident edges and edges of $R(\nu)$. We assume that C is not c-planar.

Since there exists a bijective mapping between incident edges of ν and edges of $R(\nu)$ for each cluster, the inclusion structure of T is inherited to every $R(\nu)$.

Therefore, we have to check if for each cluster ν $G(\nu)$ is drawable into the inside of $R(\nu)$ and $G - G(\nu)$ is drawable into the outside of $R(\nu)$.

We define the *inside of* $R(\nu)$ to be the bounded face of $R(\nu)$ in D and the *outside of* $R(\nu)$ to be the unbounded face in D. We assume that there is a cluster ν such that there are vertices of $G(\nu)$ and vertices of $G - G(\nu)$ drawn into the inside or outside, respectively, of $R(\nu)$. W.l.o.g. we assume that there are two components of $G(\nu)$, one drawn into the inside of $R(\nu)$, named $G_1(\nu)$, and one drawn into the outside of $R(\nu)$, called $G_2(\nu)$. Clearly,

$G_1(\nu)$ and $G_2(\nu)$ are not connected since this would result into a forbidden crossing with $R(\nu)$ in D. The same argument holds for $G - (G_1(\nu) \cup G_2(\nu))$. Furthermore, there is one connected component C_1 of $G - (G_1(\nu) \cup G_2(\nu))$ connected to $G_1(\nu)$ and one connected component C_2 of $G-(G_1(\nu) \cup G_2(\nu))$ connected to $G_2(\nu)$. Additionally, C_1 and C_2 are not connected because of the previous argument. Notice that $C_1 - G_1$ or $C_2 - G_2$, respectively, and G_2 or G_1, respectively, are not connected since this connection would have an incident edge that has to cross exactly one edge of $R(\nu)$ once. Hence, C_1 may be swapped such that G_1 lies inside of $R(\nu)$. Observe that the ordering of the incident edges of ν does not change by this modification.

Consequently, we may assume that there is a planar drawing of G such that for each cluster ν $G(\nu)$ is drawable into the inside of $R(\nu)$ and $G - G(\nu)$ into the outside of $R(\nu)$. Let us assume that we have given such a planar drawing D of G.

Since $R(\nu)$ is a cycle that corresponds to a cluster ν we may add $R(\nu)$ to cluster ν. Furthermore, we duplicate $R(\nu)$ such that it only crosses the incident edges of ν and it crosses them in the same ordering as $R(\nu)$ but lies outside of $R(\nu)$ in D. We name this added cycle $R'(\nu)$ and the set of all those R'. We do this transformation for every cluster ν in D. Since we modify D only by adding exactly the same cycle $R(\nu)$ for each cluster ν the resulting drawing has the same properties as D. Consequently, for every cluster ν we may add $R'(\nu)$ to $G - G(\nu)$. Then we add a dummy vertex on each crossing in D. We name the final graph $\hat{G}$ and its drawing $\hat{D}$. Observe that $\hat{D}$ is planar and for each cluster ν $\hat{G}(\nu)$ and $\hat{G} - \hat{G}(\nu)$ are each connected in $\hat{D}$. Then, $\hat{D}$ is c-planar and hence the corresponding clustered graph $\hat{C} = (\hat{G}, \hat{T})$ is c-planar (see [15, 54] and Chapter 8). Therefore, C is c-planar that contradicts our assumption. □

The last theorem gives us a nice equivalence: we may model a clustered graph $C = (G, T)$ by its underlying graph G and certain cycles that model the clustering structure of T. The next step is to investigate what kind of crossings we can allow in order to highlight the clustering structure of T in the crossing minimal drawing of C.

Since we are interested in an easy readability of the visualization of clustered graph we add the following constraint such that the clustering structure is highlighted in the final drawing D: a cluster region is only allowed to be crossed by incident edges of the corresponding cluster in D and no crossing is allowed between the incident edges themselves in the near of the boundary of the cluster region in D. Furthermore, we only allow crossings between edge segments of D that are drawn into the region of the same lowest common ancestor in T.

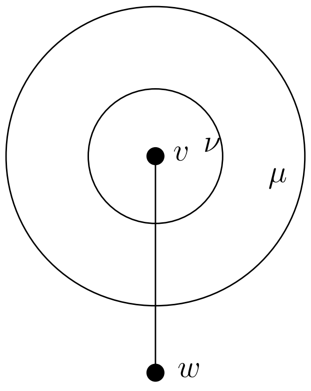
To easily handle the previous constraint we apply a pre-processing step that transforms the given clustered graph C into an auxiliary clustered graph $C_{\texttt{aux}}$ with the following properties:

- an incident edge is incident to exactly one cluster,
- two edges e_1, e_2 cross at most once that have the same lowest common ancestor in T and are no incident edges to some cluster.

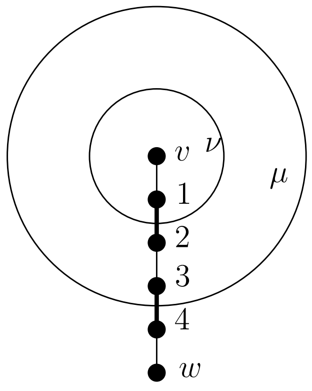
- all other edges are not allowed to cross.

Given a clustered graph $C = (G, T)$, we replace each edge $e = (v, w)$ that is incident to any cluster in the following way. First we determine the number of clusters c to whom e is incident. Then we replace e by a path $P(e)$ of length $2c + 1$.

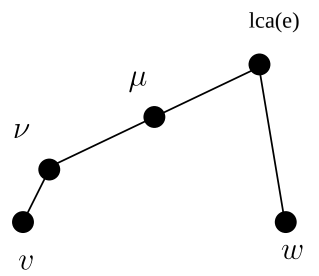
We traverse $P(e)$ and T simultaneously from v to w (see Figure 10.8). In $P(e)$ we visit every edge on an even position that we call *fixed edge*. The edges on the odd positions in $P(e)$ we name *variable edges*. In T we traverse each cluster on the unique path $P_T(e)$ from v over the lowest common ancestor of e to w.



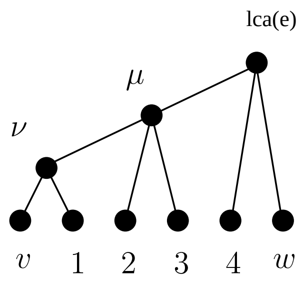
(a) An original edge $e = (v, w)$ that is incident to cluster ν and μ



(b) Edge e after transformation into path $P(e)$, bold edges are fixed edges, the others are variable edges



(c) The sub-clustered tree of $\texttt{lca}(e)$ before transformation



(d) The sub-clustered tree of $\texttt{lca}(e)$ after transformation

Figure 10.8: Transformation of incident edges of two clusters ν and μ.

Every fixed edge in $P(e)$ is assigned as incident edge to the current visited cluster ν in $P_T(e)$ and its previous variable edge in $P(e)$ is assigned to ν. Since we have c fixed edges in $P(e)$ we have a bijective mapping from $P(e)$ to $P_T(e)$. Consequently, after transformation each fixed edge is an incident edge of exactly one cluster in T.

Clearly, the resulting graph is still a clustered graph that we call *aux-clustered graph* C_{aux}

in the following. In order to get our pre-defined drawing of C_{aux} with the minimum number of crossings we set the following conditions:

1. Each fixed edge in C_{aux} is only allowed to cross the boundary of its corresponding cluster region and the number of crossings is exactly one.
2. Fixed edges are not allowed to cross each other.
3. Two edges e_1, e_2 of C_{aux} that are no fixed edges are allowed to cross each other if and only if $\mathtt{lca}(e_1) = \mathtt{lca}(e_2)$.

The lowest common ancestor of an edge can be computed in $O(1)$ time in T after a linear time pre-processing step in T [45, 76]. For each cluster ν, to model the boundary of its cluster region we add a cycle $R(\nu)$ of length the number of fixed incident edges of ν. To force the first condition we seek for a bijective mapping from the fixed incident edges of ν to the edges of the cycle $R(\nu)$ such that each assignment corresponds to a crossing between the mapped edges and fulfills our conditions. Additionally, we forbid edges of those cycles to cross each other.

In the previous subsection we have calculated the aux-clustered graph $C_{\mathsf{aux}} = (E_{\mathsf{aux}}, V_{\mathsf{aux}})$ with a set of cycles R corresponding to the boundary of the cluster regions of C_{aux}.

We apply the exact crossing minimization algorithm to C_{aux} and R: all original and variable edges of C_{aux} are replaced by a path of length (at the most) $|E_{\mathsf{aux}}|$.

From now on we may assume that every edge pair in C_{aux} and R has at the most one crossing in the final drawing. Notice that some edge pairs are not allowed to cross and some have to cross as we described in the last section.

Since omitting a variable x_{ef} for two edges e,f in the ILP is equal to setting x_{ef} to zero implicitly, we focus on edge pairs that might have or have to have a crossing in the following.

Therefore, we get the following ILP constraints:

1. For all clusters ν
   (a) for all fixed incident edges e of ν:
   $$\sum_{\hat{e} \in R(\nu)} x_{e\hat{e}} = 1.$$
   (b) for all pairs of a fixed incident edge e of ν and an edge $\hat{e}$ of $R(\nu)$
   $$x_{e\hat{e}} \in \{0, 1\}.$$
2. For a pair e, $\hat{e}$ of non-fixed edges with $\mathtt{lca}(e) = \mathtt{lca}(\hat{e})$
   $$x_{e\hat{e}} \in \{0, 1\}.$$

Notice that if we get a solution with the variables of the last item set to zero, this solution can be transformed in a c-planar drawing of C. Therefore, in this case the original clustered graph C is c-planar.

Bibliography

[1] T. Asano. An application of duality to edge-deletion problems. *SIAM Journal on Computing*, 16(2):312–331, 1987.

[2] Christian Bachmeier, Franz J. Brandenburg, and Michael Forster. Radial level planarity testing and embedding in linear time. In *Proceedings Graph Drawing 2004*, volume 2912 of *Lecture Notes in Computer Science*, pages 393–405, 2004.

[3] Guiseppe Di Battista and Fabrizio Frati. Efficient *c*-planarity testing for embedded flat clustered graphs with small faces. In *Proceedings Graph Drawing 2007*, 2008. to appear.

[4] C. Bentz, M.-C. Costa, L. Létocart, and F. Roupin. A bibliography on multicut and integer multiflow problems. Technical report, Rapport scientifique CEDRIC 654, 2004.

[5] P. Bertolazzi, G. Di Battista, and W. Didimo. Quasi-upward planarity. *Algorithmica*, 32:474–506, 2002.

[6] T. C. Biedl, M. Kaufmann, and P. Mutzel. Drawing planar partitions ii: Hh-drawings. In *Workshop on Graph Theoretic Concepts in Computer Science*, volume 1517 of *Lecture Notes in Computer Science*, pages 124–136. Springer-Verlag, 1998.

[7] H. L. Bodlaender and G. Kant. Planar graph augmentation problems. In *In Proc. 2nd Workshop Algorithms Data Struct.*, volume 519 of *Lecture Notes in Computer Science*, pages 286–298. Springer-Verlag, 1991.

[8] K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.

[9] C. Buchheim, D. Ebner, M. Jünger, G. W. Klau, P. Mutzel, and R. Weiskircher. Exact crossing minimization. In *Graph Drawing 2005*, 2006.

[10] C. Buchheim, M. Jünger, A. Menze, and M. Percan. Bimodal crossing minimization. In *Proceedings of the 12th Annual International Conference, Computing and Combinatorics, COCOON 2006*, volume 4112 of *Lecture Notes in Computer Science*, pages 497–506, August 2006.

[11] I. Cederbaum, S. Even, and A. Lempel. An algorithm for planarity testing of graphs. *In Theory of Graphs, International Symposium, Rome*, pages 215–232, 1967.

[12] Z.-Z. Chen and X. He. Hierarchical topological inference on planar disc maps. In *Proceedings of the 6th Annual International Conference on Computing and Combinatorics*, pages 115–125, July 2000.

[13] Z.-Z. Chen and X. He. Disk embeddings of planar graphs. *Algorithmica*, 38(4):539–576, 2004.

[14] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. of Computer and System Sciences*, 30(1):54–76, 1985.

[15] S. Cornelson and D. Wagner. Completely connected clustered graphs. In *Workshop on Graph Theoretic Concepts in Computer Science*, volume 2880 of *Lecture Notes in Computer Science*, pages 168–179. Springer-Verlag, 2003.

[16] P. F. Cortese, G. Di Battista, M. Patrignani, and M. Pizzonia. Clustering cycles into cycles of clusters. In *Proceedings on Graph Drawing 2004 (GD 04)*, volume 3383 of *Lecture Notes in Computer Science*, pages 100–110. Springer-Verlag, 2004.

[17] M.-C. Costa, L. Létocart, and F. Roupin. Minimal multicut and maximal integer multiflow: A survey. *EJOR European Journal on Operational Research*, 162(1):55–69, 2005.

[18] E. Dahlhaus. Linear time algorithm to recognize clustered planar graphs and its parallelization (extended abstract). In C. L. Lucchesi, editor, *LATIN '98, 3rd Latin American symposium on theoretical informatics, Campinas, Brazil, April 20–24, 1998.*, volume 1380 of *Lecture Notes in Computer Science*, pages 239–248, 1998.

[19] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[20] Elias Dahlhaus, Karsten Klein, and Petra Mutzel. Planarity testing for c-connected clustered graphs. Technical report, Department of Computer Science, University of Dortmund, June 2006. SYS-1/06, LSXI.

[21] G. Di Battista, W. Didimo, and A. Marcandalli. Planarization of clustered graphs (extended abstract). In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, volume 2265 of *Lecture Notes in Computer Science*, pages 60–74. Springer-Verlag, 2002.

[22] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.

[23] H. N. Djidjev. A linear algorithm for the maximal planar subgraph problem. In *Proceedings 4th International Workshop on Algorithms and Datastructures, WADS '95*, volume 955, pages 369–380. Springer-Verlag, Lecture Notes in Computer Science, 1995.

[24] P. Eades, Q.-W. Feng, and X. Lin. Straightline drawing algorithms for hierarchical graphs and clustered graphs. In *Graph Drawing GD 97*, volume 1190 of *Lecture Notes in Computer Science*, pages 113–128. Springer-Verlag, 1997.

[25] P. Eades, Q.-W. Feng, and H. Nagamochi. Drawing clustered graphs on an orthogonal grid. *Journal of Graph Algorithms and Applications*, 3:3–29, 2000.

[26] M. J. Egenhofer. Reasoning about binary topological relations. In *Advances in Spatial Databases*, SSD'91 Proceedings, pages 143–160. Springer-Verlag, 1991.

[27] L. Euler. Demonstratio nonnullarum insignium proprietatum quibus solida hedris planis inclusa sunt praedita. *Novi Comm. Acad. Sci. Imp. Petropol.*, 4:140–160, 1750. 1752-3, published 1758, also: Opera Omnia (1) **26**, 94–108.

[28] L. Faria, C. M. H. de Figueiredo, and C. F. X. de Mendonça N. Splitting number is NP-complete. *Discrete Applied Mathematics*, 108(1–2):65–83, 2001.

[29] Q.-W. Feng. *Algorithms for Drawing clustered graphs.* PhD thesis, Department of Computer Science and Software Engineering, University of Newcastle, April 1997. PhD Thesis.

[30] Q.-W. Feng, R.-F. Cohen, and P. Eades. Planarity for clustered graphs. In P. Spirakis, editor, *Algorithms – ESA '95, Third Annual European Symposium*, volume 979 of *Lecture Notes in Computer Science*, pages 213–226. Springer-Verlag, 1995.

[31] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16:1004–1022, 1987.

[32] H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, The Association for Computing Machinery, New York, pages 448–456, 1983.

[33] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

[34] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):312–316, 1983.

[35] N. Garg, V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997.

[36] E. Gassner and M. Percan. Maximum planar subgraph on graphs not contractive to k_5 or $k_{3,3}$. Technical report, University of Cologne, 2006. zaik2006-525.

[37] E. Gassner and M. Percan. On the weighted minimal deletion of rooted bipartite minors. Technical report, University of Cologne, 2007. zaik2007-542.

[38] Michael T. Goodrich, George S. Lueker, and Jonathan Z. Sun. c-planarity of extrovert clustered graphs. In *Proceedings Graph Drawing 2006*, pages 211–222, 2006.

[39] M. Grigni, D. Papadias, and C. H. Papadimitriou. Topological inference. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 901–906, 1995.

[40] C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher. Advances in c-planarity testing of clustered graphs. In *Proceedings on Graph Drawing 2002 (GD 02)*, volume 2528, pages 220–235. Springer-Verlag, 2002.

[41] C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher. Subgraph induced planar connectivity augmentation. In *Workshop on graph Theoretic concepts in computer science*, volume 2880, pages 261–272. Springer-Verlag, 2003.

[42] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. In J. Marks, editor, *Graph Drawing (Proc. 2000)*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer-Verlag, 2001.

[43] C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005.

[44] T. Hagerup and C. Uhrig. Triangulating a planar map without introducing multiple arcs. Technical report, 1989.

[45] D. Harel and R.Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13:338–355, 1984.

[46] R. Hassin and D. B. Johnson. An $o(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM Journal on Computing*, 14:612–624, 1985.

[47] M. R. Henzinger, Ph. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

[48] P. Hliněný. Crossing number is hard for cubic graphs. In *MCFS 2004*, pages 772–782, 2003.

[49] J. Hofcroft and R. E. Tarjan. Efficient planarity testing. *Journal of ACM*, 21(4):549–568, 1974.

[50] R. Jayakumar, K. Thulasiraman, and M. N. S. Swamy. $O(n^2)$ algorithms for graph planarization. *IEEE Transactions on Computer-Aided Design*, 8:257–267, 1989.

[51] Eva Jelinkova, Jan Kara, Jan Kratochvil, Marin Pergel, Ondrej Suchy, and Tomas Vyskocil. Clustered planarity: Small clusters in eulerian graphs. In *Proceedings Graph Drawing 2007*, 2008. to appear.

[52] L. K. Jørgensen. Vertex partitions of $k_{4,4}$-minor free graphs. *Graphs Combin.*, 17:265–274, 2001.

[53] L. K. Jørgensen and K. Kawarabayashi. Extremal results for rooted minor problems. Technical report, 2005. Report R-2005-07.

[54] M. Jünger, S. Leipert, and M. Percan. Triangulating clustered graphs. Technical report, Institut für Informatik, Universität zu Köln, 2002. zaik2002-444.

[55] M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16(1):33–59, 1996.

[56] M. Jünger and P. Mutzel. *Technical Foundations*, pages 9–49. Mathematics + Visualization. Springer-Verlag, 2004.

[57] Michael Jünger, Petra Mutzel, , Merijam Percan, and Klaus Truemper. Personal communications. 2005.

[58] K. Kuratowski. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae*, 15:271–283, 1930.

[59] J. A. La Poutré. Alpha-algorithms for incremental planarity testing. In *STOC '94*, pages 706–715, 1994.

[60] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs. Internatonal Symposium, Rome, Italy, July 1966*, pages 215–232. Gordon and Breach, New York, 1967.

[61] T. Lengauer. Hierarchical planarity testing algorithms. *Journal of the Association for Computing Machinery*, 36(3):474–509, 1989.

[62] K.-F. Liao and M. Sarrafzadeh. Vertex-disjoint trees and boundary single-layer routing. In R. H. Möhring, editor, *Proceedings 16th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '90)*, volume 484 of *Lecture Notes in Computer Science*, pages 99–108, 1990.

[63] A. Liebers. Planarizing graphs – a survey and annotated bibliography. *Journal of Graph Algorithms and Applications*, 5(1):1–74, 2001.

[64] P. C. Liu and R. C. Geldmacher. On the deletion of nonplanar edges of a graph. In *Proc. of the 10th Southeastern Conference on Combinatorics, Graph Theory, and Computing, Boca Raton, Florida, USA, 1979, part 2*, volume 24, pages 727–738. Congressus Numerantium, 1979.

[65] J. F. Lynch. The equivalence of theorem proving and the interconnection problem. *(ACM) SIGDA Newsletter*, 5(3):31–36, 1975.

[66] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16:233–242, 1996.

[67] A. Menze. Darstellung von Nebenmetaboliten in automatisch erzeugten Zeichnungen metabolischer Netzwerke. Master's thesis, Institute of Biochemistry, University of Cologne, June 2004.

[68] H. Nagamochi and K. Kuroya. Convex drawing for c-planar biconnected clustered graphs. In *Proceedings on Graph Drawing 2003 (GD 03)*, volume 2912, pages 369–380. Springer-Verlag, 2004.

[69] M. J. Pelsmajer, M. Schaefer, and D. Štefankovič. Crossing number of graphs with rotation systems. Technical report, Department of Computer Science, DePaul University, 2005.

[70] R.-C. Read. A new method for drawing a planar graph given the cyclic order of the edges at each vertex. *Congressus Numerantium*, 56:31–44, 1987.

[71] N. Robertson and P. Seymour. An outline of a disjoint paths algorithm. In *Paths, Flows, and VLSI-Layout*, pages 267–292, 1990.

[72] N. Robertson, P. Seymour, and R. Thomas. Hadwiger's conjecture for K_6-free graphs. *Combinatorica*, 13(3):279–361, 1993.

[73] N. Robertson and P. D. Seymour. Graph minors XIII. the disjoint paths problem. *Journal of Combinatorial Theory Series B*, 63:65–110, 1995.

[74] Rome library of directed graphs.
http://www.inf.uniroma3.it/people/gdb/wp12/directed-acyclic-1.tar.gz.

[75] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Symposium on the Theory of Computing*, pages 216–226, 1978.

[76] B. Schieber and U. Vishkin. On finding lowest common ancestors, simplification and parallelization. *SIAM Journal on Computing*, 17:1253–1262, 1988.

[77] R. Smith and K. K. Park. Algebraic approach to spatial reasoning. *International Journal Geographical Information Systems*, 6:177–192, 1992.

[78] K. Sugiyama and K. Misue. Visualization of structural information: automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 4(21):876–893, 1991.

[79] H. Suzuki, T. Akama, and T. Nishizeki. Finding Steiner forests in planar graphs. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 90)*, pages 444–453, 1990.

[80] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[81] K. Truemper. *Matroid Decomposition.* Academic Press, University of Texas at Dallas, Richardson, Texas, 1992.

[82] K. Truemper. Personal communications. July 2006.

[83] D. Wagner. Simple algorithms for Steiner trees and paths packing problems in planar graphs. *CWI Quarterly*, 6(3):219–240, 1993.

[84] K. Wagner. Über eine Eigenschaft der ebenen Komplexe. *Mathematische Annalen*, 114:570–590, 1937.

[85] T. Watanabe, T. Ae, and A. Nakamura. On the NP-hardness of edge-deletion and -contraction problems. *Discrete Applied Mathematics*, 6:63–78, 1983.

[86] M. Yannakakis. Node- and edge-deletion NP-complete problems. In *STOC '78*, pages 253–264, 1978.

[87] M. Yannakakis. Edge-deletion problems. *SIAM Journal on Computing*, 10:297–309, 1981.

Index

Ich versichere, daß ich die von mir vorgelegte Dissertation selbständig und ohne unzulässige Hilfe angefertigt, die benutzen Quellen und Hilfsmittel vollständig angegeben und die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken im Wortlaut oder dem Sinn nach entnommen sind, in jedem Einzelfall als Entlehnung kenntlich gemacht habe; daß diese Dissertation noch keiner anderen Fakultät zur Prüfung vorgelegen hat; daß sie abgesehen von unten angegebenen Teilpublikationen noch nicht veröffentlicht ist, sowie daß ich eine solche Veröffentlichung vor Abschluß des Promotionsverfahrens nicht vornehmen werde. Die Bestimmungen der geltenden Promotionsordnung sind mir bekannt. Die von mir vorgelegte Dissertation ist von Professor Dr. Michael Jünger betreut worden.

Köln, im November 2007 ______________________________

(Merijam Percan)

- C. Buchheim, M. Jünger, A. Menze, and M. Percan.
  Bimodal crossing minimization.
  In *Proceedings of the 12th Annual International Conference, Computing and Combinatorics, COCOON 2006*, volume 4112 of *Lecture Notes in Computer Science*, pages 497–506, August 2006.

- E. Gassner and M. Percan.
  Maximum planar subgraph on graphs not contractive to K_5 or $K_{3,3}$.
  Technical report, University of Cologne, 2006.
  zaik2006-525.

- E. Gassner and M. Percan.
  On the weighted minimal deletion of rooted bipartite minors.
  Technical report, University of Cologne, 2007.
  zaik2007-542.

- C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher.
  Advances in c-planarity testing of clustered graphs.
  In *Proceedings on Graph Drawing 2002 (GD 02)*, volume 2528, pages 220–235. Springer-Verlag, 2002.

- C. Gutwenger, M. Jünger, S. Leipert, P. Mutzel, M. Percan, and R. Weiskircher.
  Subgraph induced planar connectivity augmentation.
  In *Workshop on graph Theoretic concepts in computer science*, volume 2880, pages 261–272. Springer-Verlag, 2003.

- M. Jünger, S. Leipert, and M. Percan.
  Triangulating clustered graphs.
  Technical report, Institut für Informatik, Universität zu Köln, 2002. zaik2002-444.