technische universität
dortmund

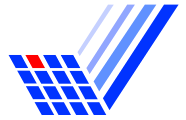
Algorithmen zur Bestimmung von guten Graph-Einbettungen für orthogonale Zeichnungen

Thorsten Kerkhof

Algorithm Engineering Report
TR07-1-011
November 2007
ISSN 1864-4503

Fakultät für Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK



Thorsten Kerkhof

Algorithmen zur Bestimmung von guten Graph-Einbettungen für orthogonale Zeichnungen

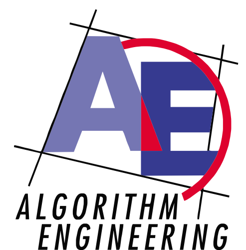
Diplomarbeit

16. August 2007

INTERNE BERICHTE
INTERNAL REPORTS

Lehrstuhl für Algorithm Engineering (LS11)
Otto-Hahn-Str. 14
44227 Dortmund

Gutachter:
Prof. Dr. Petra Mutzel
Dipl.-Inform. Carsten Gutwenger



GERMANY · D-44221 DORTMUND

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die Diplomarbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, sowie Zitate kenntlich gemacht habe.

Dortmund, den 16. August 2007 ______________________________

(Thorsten Kerkhof)

Inhaltsverzeichnis

1 Einführung 1
1.1 Motivation 4
1.2 Ziele 4
1.3 Gliederung der Diplomarbeit 5

2 Grundlagen der Graphentheorie 7
2.1 Grundlegende Begriffe 7
2.2 Topology-Shape-Metrics Ansatz 11
2.3 SPQR-Bäume 13
2.4 BC-Bäume 20

3 Berechnung einer planaren Einbettung für planare Graphen 25
3.1 Minimale Tiefe Algorithmus von Pizzonia und Tamassia 27
3.1.1 Experimentelle Analyse 31
3.1.2 Modifikation für minimale erweiterte Tiefe 34
3.2 Maximale Außenfläche 36
3.2.1 Zweizusammenhängende Graphen 36
3.2.2 Zusammenhängende Graphen 43
3.3 Minimale erweiterte Tiefe 47
3.3.1 Modifikation des Algorithmus für minimale Tiefe 49
3.4 Minimale Tiefe und Maximale Außenfläche 52

4 Einbettung von inneren Blöcken 53
4.1 Simpler Ansatz 53
4.2 Kleine Flächen 54
4.3 Wenige große Flächen 57
4.4 Große äußere Schichten 59
4.4.1 Zweizusammenhangskomponenten 60
4.4.2 Zusammenhängende Graphen 70

5 Experimentelle Vergleiche 73
5.1 Minimale Tiefe und erweiterte minimale Tiefe 75
5.2 Algorithmen zur Berechnung von planaren Einbettungen 79
5.3 Methoden für die Einbettung von inneren Blöcken 83
5.3.1 Varianten im Vergleich untereinander 84
5.3.2 Beste Varianten im Vergleich mit dem Rest 93

6 Zusammenfassung und Ausblick 97
6.1 Zusammenfassung 97
6.2 Ausblick 99

A Beiliegende DVD 101

Literaturverzeichnis 103

Abbildungsverzeichnis 107

1 Einführung

Graphen finden in vielen wissenschaftlichen und wirtschaftlichen Gebieten Anwendung. Dabei werden Zeichnungen für Graphen, in denen Knoten als Kästen und Kanten als Sequenz von senkrechten und waagerechten Liniensegmenten dargestellt werden, so genannte orthogonale Zeichnungen, in vielen Anwendungsgebieten bevorzugt. Orthogonale Zeichnungen werden z.B. häufig für Entity-Relationship Diagramme [Che76], UML Diagramme [RJB04] (siehe Abbildung 1.1) oder die Visualisierung von Datenbanken benutzt. Ein beliebter Ansatz, um orthogonale Zeichnungen für Graphen zu erstellen, ist der Topology-Shape-Metrics Ansatz [BE+98, TBB88]. Die Güte der Zeichnung hängt dabei stark davon ab, wie gut die planare Einbettung ist, die während der Anwendung des Ansatzes berechnet wird. Eine planare Einbettung legt die Gestalt des Graphen fest, die Koordinaten der Knoten und Kantenstützstellen werden jedoch in einem weiteren Schritt berechnet. In den Abbildungen 1.2 und 1.3 sind jeweils zwei verschiedene Zeichnungen für den selben Graphen erstellt worden, wobei sich der Erstellungsprozess nur in der Berechnung der planaren Einbettung unterscheidet. Es ist ein deutlicher Unterschied in der Güte der Zeichnungen zu erkennen, man würde jeweils die zweite Zeichnung der ersten vorziehen. Diese Diplomarbeit beschäftigt sich mit Algorithmen für die Berechnung einer planaren Einbettung nach bestimmten Kriterien.

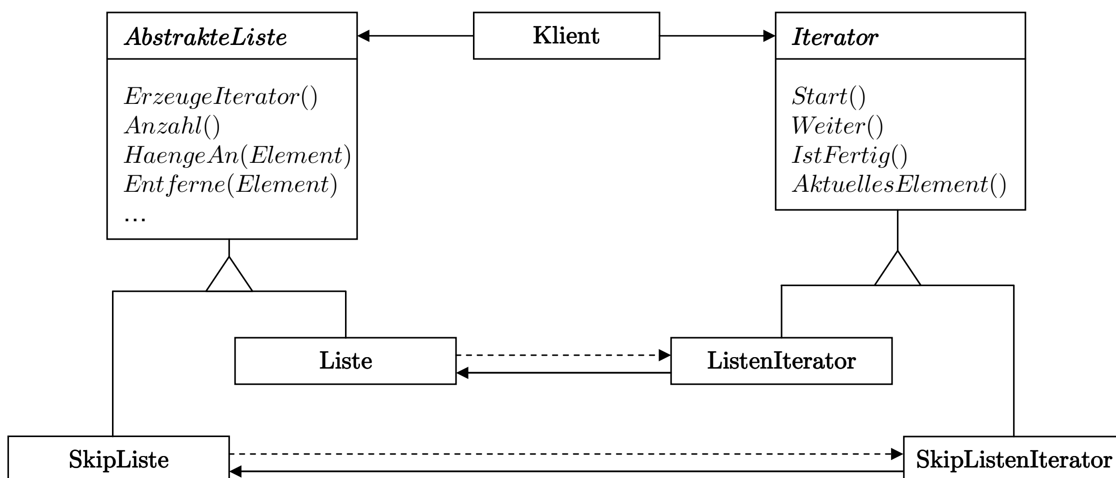
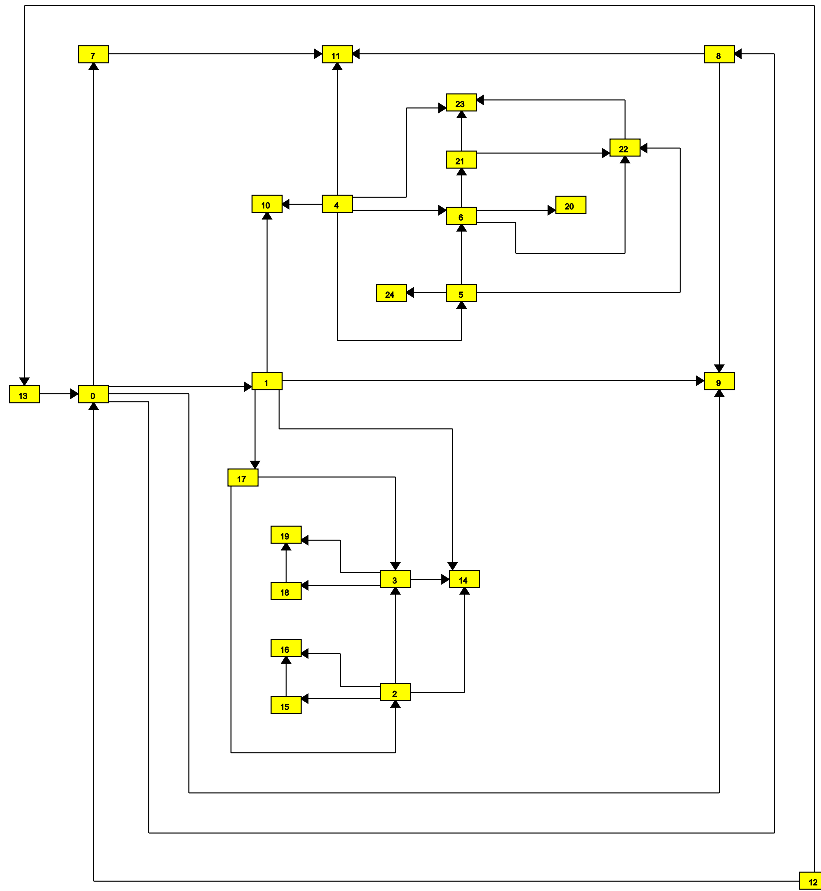
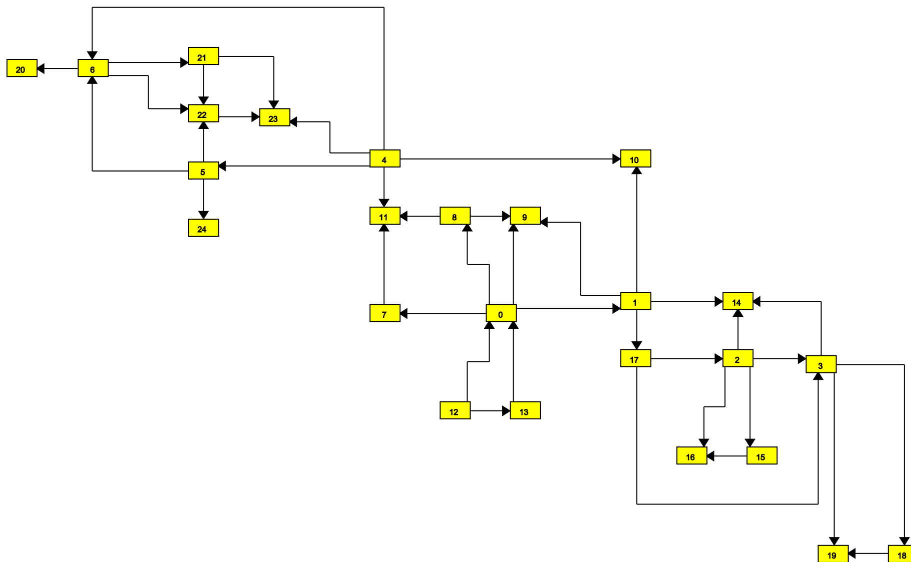


Abbildung 1.1: Beispiel für ein UML Klassendiagramm. (Quelle: [GH+97])

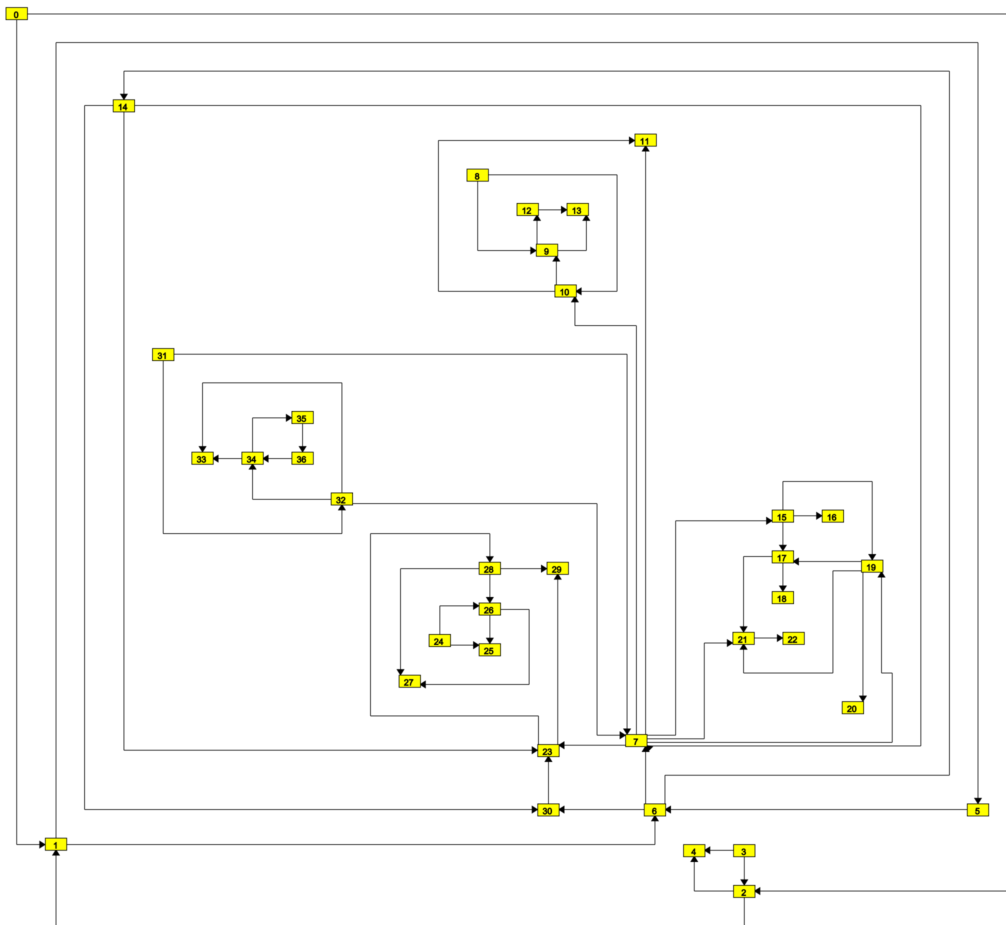


(a) Zeichnung für den Beispielgraphen mit planarer Einbettung (Γ_1, f_1)

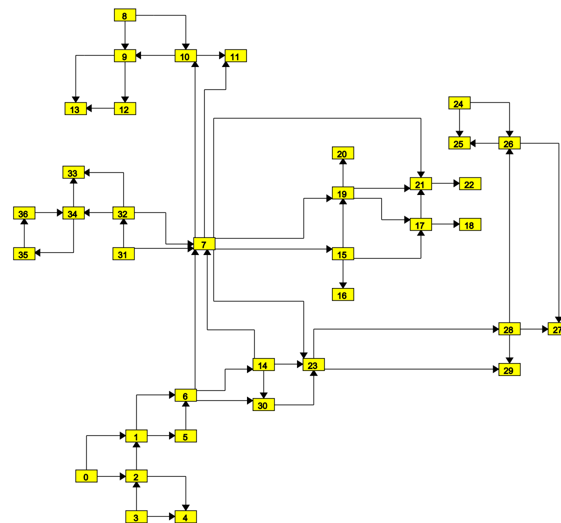


(b) Zeichnung für den Beispielgraphen mit planarer Einbettung (Γ_2, f_2)

Abbildung 1.2: Einfluss der planaren Einbettung auf die Zeichnung für einen Beispielgraphen mit 23 Knoten.



(a) Zeichnung für den Beispielgraphen mit planarer Einbettung (Γ_1, f_1)



(b) Zeichnung für den Beispielgraphen mit planarer Einbettung (Γ_2, f_2)

Abbildung 1.3: Einfluss der planaren Einbettung auf die Zeichnung für einen Beispielgraphen mit 36 Knoten.

1.1 Motivation

Für Graphen mit maximalem Knotengrad vier kann, falls eine planare Einbettung des Graphen gegeben ist, eine planare orthogonale Zeichnung mit minimaler Kantenknickanzahl erstellt werden [Tam87] und für Graphen mit höherem maximalem Knotengrad existieren Erweiterungen dieses Algorithmus, wie z.B. [FK96]. Falls die planare Einbettung des Graphen nicht gegeben ist, ist das Problem, eine knickminimale Zeichnung zu finden, in bestimmtem Fällen selbst für Graphen mit maximalem Knotengrad vier, NP-schwer. Das Problem ist unter anderem dadurch schwierig, da ein Graph exponentiell viele planare Einbettungen besitzen kann.

Es gibt Branch-and-Cut Verfahren [LVB94, MW02], um eine exakte Lösung, das heißt eine knickminimale Zeichnung, zu finden. Aufgrund der exponentiellen Worst-Case Rechenzeit eignen sie sich jedoch nur für relativ kleine Graphen mit bis zu 80 Knoten. Das Ziel ist es daher, einen Algorithmus zu entwickeln, welcher effizient eine „gute“ planare Einbettung berechnet. Der Algorithmus wird im Topology-Shape-Metrics Ansatz genutzt und die berechnete Einbettung dient als Eingabe für eines der effizienten Knickminimierungsverfahren (z.B. [FK96]).

1.2 Ziele

Bei der Berechnung einer planaren Einbettung gibt es verschiedene Einbettungsmaße, welche optimiert werden können. Es existieren bereits Verfahren, um die maximale topologische Verschachtelungstiefe zu minimieren [PT00, GM04b], die Anzahl der Kanten in der Außenfläche zu maximieren [GM04b] oder über alle Einbettungen mit minimaler Verschachtelungstiefe eine Einbettung mit maximaler Außenfläche zu bestimmen [GM04b]. Eine der Hauptaufgaben der Diplomarbeit ist es, diese Algorithmen zu implementieren und miteinander zu vergleichen. Zeichnungen mit minimaler Knickanzahl sind in der Regel auch gut in Hinsicht auf andere Bewertungskriterien. Da dies aber nicht immer der Fall sein muss, wird untersucht, wie sich die Verwendung der Algorithmen für alternative Einbettungsmaße auf unterschiedliche ästhetische Kriterien auswirkt. Beispiele für andere Bewertungskriterien als die Knickanzahl sind die Größe der Zeichenfläche und die Summe der Kantenlängen. Der Algorithmus für minimale topologische Verschachtelungstiefe, die so genannte minimale Tiefe, von Pizzonia und Tamassia wurde bereits 2005 von Pizzonia implementiert und mit einem Algorithmus, der eine Einbettung mit großer Tiefe berechnet, verglichen [Piz05]. In den Experimenten wurde festgestellt, dass minimale Tiefe einen Vorteil gegenüber großer Tiefe für die Kriterien Knickanzahl, Größe der Zeichenfläche und Summe der Kantenlängen hat.

Dieser minimale Tiefe Algorithmus besitzt allerdings eine Einschränkung: Die Einbettung der Zweizusammenhangskomponenten des Graphen muss vorgegeben sein. Gutwenger und Mutzel präsentieren in [GM04b] jedoch einen Algorithmus für minimale Tiefe, welcher für allgemeine Graphen funktioniert. In der Diplomarbeit wird untersucht, wie sich die Einschränkung des Algorithmus von Pizzonia und Tamassia auf die Güte der entstehenden Zeichnungen auswirkt. Es wird auch ein

neues Bewertungskriterium, die Größe der inneren Flächen, zusätzlich zu den in [Piz05] vorgestellten Kriterien eingeführt, welches ein angemessenes Maß über die Güte einer Zeichnung gibt.

Wie bereits erwähnt, neigt man dazu, die Zeichnung in Abbildung 1.2(b) der in Abbildung 1.2(a) vorzuziehen. Betrachtet man die vorgestellten Bewertungskriterien, so ergibt sich dort auch ein klarer Vorteil der Zeichnung aus Abbildung 1.2(b). Ihre Kantenknickanzahl ist mit 20 im Vergleich zu 30 deutlich geringer, die Größe der Zeichenfläche ist um ca. 29 Prozent kleiner und die Summe der Kantenlänge ist sogar um ca. 60 Prozent geringer. Genauso ergeben sich bei der Bewertung anhand dieser Kriterien klare Vorteile für die zweite Zeichnung in Abbildung 1.3, dort ist die Kantenknickanzahl in Abbildung 1.3(b) um 28 geringer als die in Abbildung 1.3(a), die Größe der Zeichenfläche um ca. 58 Prozent kleiner und die Summe der Kantenlänge um ca. 72 Prozent geringer. Das unterstreicht, dass diese Bewertungskriterien Aufschluss über die subjektiv empfundene Güte geben.

Am Lehrstuhl für Algorithm Engineering der Universität Dortmund wird eine Algorithmenbibliothek, das so genannte Open Graph Drawing Framework (OGDF), verwendet. In dieses werden die Implementierungen aller behandelten Algorithmen integriert. Im OGDF existieren bereits viele Algorithmen und Datenstrukturen, die benötigt werden, wie z.B. eine Implementierung für Knickminimierungsverfahren, BC- und SPQR-Bäume.

Weiterhin wird erarbeitet, wie sich das Erstellen einer Einbettung noch weiter verbessern lässt. Ein Ansatzpunkt ist die Tatsache, dass die Algorithmen aus [PT00, GM04b] nur bestimmen, wie die Außenfläche des Graphen und die Verschachtelung von Zweizusammenhangskomponenten aussieht. Die Einbettung von Kanten, die nicht in der Außenfläche liegen, wird jedoch nicht festgelegt. Daher werden drei neue Einbettungsmaße eingeführt und für diese effiziente Algorithmen entwickelt und implementiert. Die neu entwickelten Algorithmen werden dann ebenfalls experimentell, zum einen untereinander, zum anderen gegen die Algorithmen aus [PT00, GM04b], verglichen und ihre Implementierung in das OGDF integriert.

1.3 Gliederung der Diplomarbeit

In Kapitel 2 werden zunächst die Grundlagen geschaffen, welche benötigt werden, um die danach folgenden Kapitel zu verstehen. Es werden alle notwendigen Begriffe der Graphentheorie definiert und veranschaulicht und der Topology-Shape-Metrics Ansatz, welcher große Bedeutung für die Nutzung der in dieser Diplomarbeit behandelten Algorithmen hat, näher erläutert. Am Ende des Kapitels werden die Datenstrukturen der SPQR- und BC-Bäume definiert und an Beispielen erklärt.

Kapitel 3 fasst die Ergebnisse der beiden Paper von Pizzonia und Tamassia [PT00] sowie von Gutwenger und Mutzel [GM04b] zusammen. Dabei wird besonderer Wert darauf gelegt, die Algorithmen zu beschreiben, und weniger, Korrektheits- oder Laufzeitbeweise zu führen. Diese Beweise wurden bereits in den Papern selber geführt und sind für die Ziele der Diplomarbeit unerheblich. Die Beschreibung der Algo-

rithmen wird an einigen Stellen um Details ergänzt, die bei der Implementierung von Belang sind. Weiterhin werden die Ergebnisse der experimentellen Analyse von Pizzonias und Tamassias minimale Tiefe Algorithmus aus [Piz05] zusammengefasst.

Das Kapitel 4 beschreibt drei Varianten, die sich mit der Einbettung von so genannten inneren Blöcken befassen. Ein Block ist eine Zweizusammenhangskomponente eines Graphen und wird in den Algorithmen aus [PT00, GM04b] besonders behandelt. Zwei der Varianten beschäftigen sich damit, in welche Flächen Blöcke eingebettet werden sollen, die nicht in die Außenfläche eingebettet werden können. Die dritte vorgestellte Variante verändert zusätzlich noch die Einbettung der Blöcke, um das dort neu eingeführte Maß der lexikografisch maximalen Schichtenmenge zu optimieren.

Alle vorgestellten Algorithmen und Varianten wurden im Rahmen der Diplomarbeit implementiert und werden in Kapitel 5 experimentell miteinander verglichen. Das Kapitel 6 fasst die Resultate der Diplomarbeit noch einmal zusammen und gibt einen Ausblick darüber, welche weiteren Forschungen sich aus den Ergebnissen ergeben. Der Diplomarbeit liegt eine DVD bei. Der Inhalt dieser DVD ist in Anhang A beschrieben.

2 Grundlagen der Graphentheorie

In diesem Kapitel werden alle graphentheoretischen Grundlagen geschaffen, die für das Verständnis der folgenden Kapitel notwendig sind. Da viele Bücher in der Fachliteratur in englischer Sprache verfasst sind, wird bei der Einführung von Begriffen meist auch die englische Bezeichnung mit angegeben.

2.1 Grundlegende Begriffe

Bevor spezielle Methoden der Graphentheorie erläutert werden können, werden hier die grundlegenden Begriffe erklärt. Dies dient unter anderem dazu, Unklarheiten in der Notation zu vermeiden, da sich diese in der Literatur (z.B. [Har69, BM76, Cha85, BLW86, Jun94, Tur04, Die05]) teilweise stark unterscheiden.

Definition 2.1 (Graph, Teilgraph) *Ein Graph $G = (V, E)$ besteht aus einer Menge von Knoten (engl.: node oder vertex) $v \in V$ und einer Menge von Kanten (engl.: edge) $(u, v) \in E$ bzw. $\{u, v\} \in E$ mit $u, v \in V$. Man unterscheidet zwischen gerichteten und ungerichteten Kanten. Gerichtete Kanten werden mit $(u, v) \in E$ und ungerichtete mit $\{u, v\} \in E$ notiert.*

Ein Teilgraph $G' = (V', E')$ eines Graphen $G = (V, E)$ besteht aus einer Teilmenge $V' \subseteq V$ der Knoten von G und einer Teilmenge der Kanten $E' \subseteq E$.

Im folgenden, falls nicht anders angegeben, wird immer ein ungerichteter Graph $G = (V, E)$ angenommen.

Definition 2.2 (Inzidenz und Adjazenz) *Eine Kante ist inzident zu ihrem Anfangs- sowie Endknoten und umgekehrt ist ein Knoten inzident zu allen Kanten, welche diesen Knoten als Anfangs- oder Endknoten enthalten. Zwei Knoten sind adjazent, falls eine Kante zwischen ihnen existiert. Zwei Kanten sind adjazent zueinander, falls sie einen gemeinsamen Knoten haben.*

Definition 2.3 (Weg, Pfad und Kreis) *Sei $G = (V, E)$ ein gerichteter oder ungerichteter Graph. Ein Pfad (engl.: path) P der Länge k in G ist eine Sequenz der Form $P = (v_1, e_1, v_2, e_2, \ldots, e_k, v_{k+1})$ mit $e_i = (v_i, v_{i+1}) \in E$ für $i = 1, \ldots, k$, falls die Kanten in G gerichtet sind, und $e_i = \{v_i, v_{i+1}\} \in E$ für $i = 1, \ldots, k$ sonst. Alle Knoten v_1 bis v_{k+1} sind paarweise verschieden. Häufig wird ein Pfad auch nur durch $P = (v_1, v_2, \ldots, v_{k+1})$ oder $P = (e_1, e_2, \ldots, e_k)$ beschrieben.*

Ein Weg $u \to v$ zwischen zwei Knoten u und v ist ein Pfad P, wobei die Richtung der Kanten nicht betrachtet wird. Das heißt, falls G gerichtet ist, darf der Weg

Kanten in verdrehter Richtung benutzen, z.B. $(u, e_0, v_0, e_1, v_1, e_2, v)$ *mit* $e_0 = (u, v_0)$, $e_1 = (v_1, v_0)$, $e_2 = (v_1, v)$. *Ein Kreis (engl.: cycle) besteht aus der Kante* (v, u) *und dem Weg* $u \rightarrow v$.

Definition 2.4 (Zusammenhängende Graphen) *Ein Graph heißt zusammenhängend (engl.: connected), wenn für alle Knoten* $u, v \in V$ *gilt, dass* v *von* u *aus erreichbar ist, das heißt es existiert ein Weg* $u \rightarrow v$.

So lange nichts anderes angegeben ist, wird im folgenden immer von zusammenhängenden Graphen ausgegangen.

Definition 2.5 (Multigraph und Self Loop) *Ein Multigraph ist ein Graph* $G = (V, E)$, *welcher eine beliebige Anzahl Kanten zwischen zwei Knoten enthalten kann. Das bedeutet, dass* E *eine Multimenge ist, da sie mehrere Instanzen der gleichen Knotenpaare enthalten kann. Ein Self Loop ist eine Kante der Art* (v, v) *bzw.* $\{v, v\}$ *für einen Knoten* $v \in V$.

Definition 2.6 (Zwei-, drei- und *k*-zusammenhängend) *G ist zweizusammenhängend (engl.: biconnected), wenn für alle Knoten* $v \in V$ *gilt, dass* $G' = (V \setminus \{v\}, E')$ *mit* $E' = \{e \in E \mid v \notin e\}$ *zusammenhängend ist. G heißt dreizusammenhängend (engl.: triconnected), wenn für alle Knoten* $u, v \in V$ *gilt, dass* $G' = (V \setminus \{u, v\}, E')$ *mit* $E' = \{e \in E \mid u \notin e, v \notin e\}$ *zusammenhängend ist.*

Allgemein wird ein Graphen k*-zusammenhängend genannt, wenn der Graph mehr als* k *Knoten enthält,* k *beliebige Knoten aus der Knotenmenge entfernt werden können und der Restgraph zusammenhängend ist.*

Alternativ lässt sich k-zusammenhängend folgendermaßen definieren: Ein Graph ist genau dann k-zusammenhängend, wenn es zwischen jedem Paar von Knoten k unabhängige Wege gibt. Unabhängig bedeutet, dass die Wege keine internen Knoten gemeinsam haben.

Definition 2.7 (Schnittknoten) *Sei* $G = (V, E)$ *ein zusammenhängender Graph. Ein Schnittknoten (engl.: cut vertex)* $v \in V$ *ist ein Knoten, dessen Entfernen dazu führt, dass der Restgraph nicht mehr zusammenhängend ist.*

Eine alternative Definition für zweizusammenhängend ist, dass G zusammenhängend ist und keinen Schnittknoten enthält.

Definition 2.8 (Separationspaar) *Sei* $G = (V, E)$ *ein zweizusammenhängender Graph und* $\{u, v\}$ *ein Knotenpaar mit* $u \neq v$. *Falls das Entfernen der Knoten* u *und* v *dazu führt, dass* G *nicht mehr zusammenhängend ist, bezeichnet man* $\{u, v\}$ *als Separationspaar.*

Definition 2.9 (Block) *Ein Block* B *ist ein zweizusammenhängender Teilgraph von* G, *welcher maximale Größe hat. Das heißt, fügt man einen weiteren Knoten zu* B *hinzu, ist* B *nicht mehr zweizusammenhängend.*

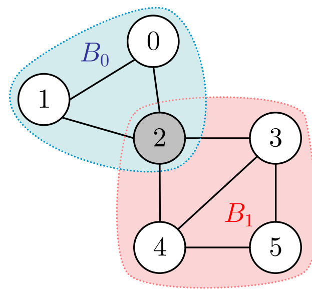


Abbildung 2.1: Der Knoten 2 ist ein Schnittknoten, B_0 und B_1 sind die Blöcke des Graphen.

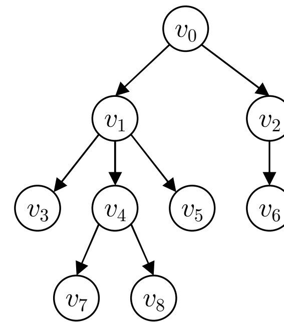


Abbildung 2.2: Baum mit Wurzel v_0.

Abbildung 2.1 veranschaulicht die Begriffe Schnittknoten und Blöcke anhand eines Beispielgraphen. Eine Spezialform eines Graphen sind so genannte Bäume:

Definition 2.10 (Baum) *Ein Baum (engl.: tree) $T = (V, E)$ ist ein gerichteter, zusammenhängender Graph mit einem ausgezeichneten Knoten $r \in V$, der Wurzel (engl.: root) von T. Der zugehörige ungerichtete Graph ist kreisfrei und die Wurzel ist nur zu Kanten mit r als Quelle inzident. Alle Knoten des Baums ohne ausgehende Kanten werden Blätter (engl.: leaf) genannt. Bäume lassen sich in Ebenen unterteilen. Die Wurzel befindet sich auf Ebene 0, alle Knoten, mit denen die Wurzel verbunden ist, sind auf Ebene 1 usw. Sind ein Knoten u auf Ebene x und ein Knoten v auf Ebene $x + 1$ miteinander verbunden, nennt man u den Elter (engl.: parent) von v und v ein Kind (engl.: child) von u. Die Kanten sind immer vom Elter zum Kind gerichtet.*

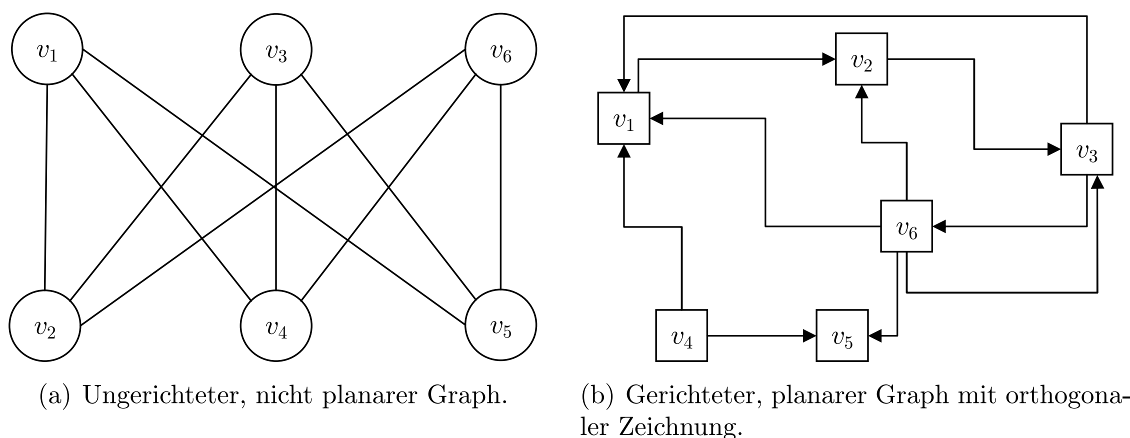
In Abbildung 2.2 ist ein Beispiel für einen Baum abgebildet.

Definition 2.11 (Tiefe eines Baums) *Als Tiefe (engl.: depth) eines Baums bezeichnet man die Länge des längsten Pfades von der Wurzel zu einem Blatt.*

Definition 2.12 (Durchmesser eines Baums) *Der Durchmesser (engl.: diameter) eines Baums T ist der längste Weg zwischen zwei beliebigen Knoten von T.*

Graphen besitzen die Eigenschaft, dass sie grafisch dargestellt werden können. Knoten werden häufig durch Kreise oder Rechtecke dargestellt und durch Linien verbunden, wenn zwischen ihnen eine Kante existiert (siehe Abbildung 2.3). Die bisherigen Definitionen bezogen sich alle auf allgemeine Eigenschaften von Graphen, die folgenden Definitionen wiederum beschäftigen sich mit den Zeichnungen eines Graphen.

Definition 2.13 (Planare Graphen) *Ein Graph heißt planar, wenn es eine Zeichnung von G gibt, so dass sich keine Kanten schneiden. Solche Überschneidungen von Kanten nennt man Kantenkreuzungen (engl.: edge crossing).*



(a) Ungerichteter, nicht planarer Graph.

(b) Gerichteter, planarer Graph mit orthogonaler Zeichnung.

Abbildung 2.3: Graphen

Definition 2.14 (Orthogonale Darstellungen) *Eine Zeichnung eines Graphen wird als orthogonal bezeichnet, falls für alle Kanten $e \in E$ gilt, dass ihre Darstellung nur aus Segmenten in horizontaler und vertikaler Richtung besteht. In orthogonalen Zeichnungen bezeichnet man einen Wechsel von einem horizontalen zu einen vertikalem Segment — oder umgekehrt — als einen Kantenknick oder kurz Knick (engl.: bend).*

In Abbildung 2.3 werden diese Begriffe veranschaulicht.

Definition 2.15 (Einbettung eines Graphen) *Eine (kombinatorische) Einbettung (engl.: (combinatorial) embedding) Γ von G ist eine Äquivalenzklasse[1] von planaren Zeichnungen für G. Für jeden Knoten werden im Uhrzeigersinn seine inzidenten Kanten abgespeichert. Ist diese Reihenfolge für zwei Zeichnungen gleich, so sind sie in der selben Äquivalenzklasse. Dabei ist nicht wichtig, mit welcher inzidenten Kante die Reihenfolge beginnt ($\{e_1, e_2, e_3\} \equiv \{e_2, e_3, e_1\}$).*

Definition 2.16 (Flächen) *In einer Zeichnung eines Graphen gibt es Regionen, die von Kanten begrenzt werden. Diese werden als Flächen (engl.: face) bezeichnet. Eine Fläche wird durch die Reihenfolge ihrer adjazenten Kanten eindeutig beschrieben. Dabei ist nicht wichtig, mit welcher Kante die Reihenfolge beginnt.*

Definition 2.17 (Länge des Außenflächenkreises, Brücken) *Jede Zeichnung hat eine ausgezeichnete Fläche, welche Außenfläche genannt wird. Diese Fläche wird nach außen durch keine Kanten beschränkt, lediglich nach innen.*

Die Länge des Außenflächenkreises, auch Größe der Außenfläche genannt, ist die Anzahl der Kanten, die diese Fläche definieren. Dabei werden so genannte Brücken doppelt gezählt. Eine Brücke nennt man die einzige Kante in einem Block, welcher aus genau zwei Knoten und dieser Kante besteht.

[1] Drückt die Ähnlichkeit von Objekten aus. In einer Äquivalenzklasse gilt für eine Relation $\sim$ Reflexivität ($a \sim a$), Symmetrie ($a \sim b \Leftrightarrow b \sim a$) und Transitivität ($a \sim b$ und $b \sim c \Rightarrow a \sim c$)

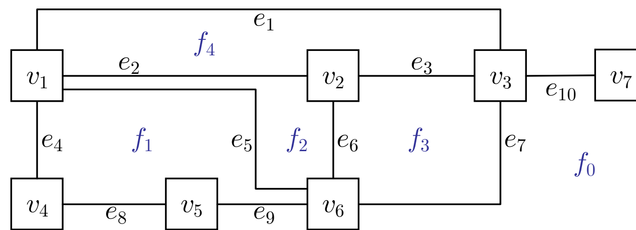


Abbildung 2.4: Flächen der Zeichnung eines Graphen.

Eine alternative Definition für eine kombinatorische Einbettung ist die Definition der Flächen samt der Ordnung ihrer adjazenten Kanten im Uhrzeigersinn. In Abbildung 2.4 ist eine Zeichnung eines Graphen samt seiner Flächen dargestellt. Fläche f_1 wird z.B. durch seine adjazenten Kanten $\{e_4, e_5, e_9, e_8\}$ definiert. Die Außenfläche der Zeichnung ist f_0 mit Größe 7. Die Kante e_{10} ist eine Brücke.

Definition 2.18 (Planare Einbettung) *Eine kombinatorische Einbettung Γ zusammen mit einer Außenfläche $f \in \Gamma$ wird planare Einbettung (Γ, f) genannt.*

Eine planare Einbettung legt die Gestalt einer Zeichnung fest.

Definition 2.19 (Dualer Graph) *Der duale Graph $G' = (V', E')$ einer kombinatorischen Einbettung Γ von G wird folgendermaßen erzeugt. Die Menge der Knoten ist $V' = F$, die Menge der Flächen von Γ. Zwischen zwei Flächen f_1 und f_2 mit $f_1 \neq f_2$ existiert genau eine Kante $\{f_1, f_2\} \in E'$, falls die Flächen mindestens eine gemeinsame Kante $e \in E$ besitzen. (vergl. Abbildung 2.13(c) und Abbildung 2.13(d))*

2.2 Topology-Shape-Metrics Ansatz

Eine sehr verbreitete und oft genutzte Technik für das Erstellen von orthogonalen Zeichnungen für allgemeine Graphen ist der Topology-Shape-Metrics Ansatz [BE+98, TBB88]. Er besteht aus drei Phasen:

1. Planarisierung
2. Orthogonalisierung
3. Kompaktierung

Im ersten Schritt wird die Topologie des Graphen berechnet. Dazu wird der Graph auf Planarität überprüft. Falls der Graph nicht planar ist, werden Kantenkreuzungen durch Dummy-Knoten ersetzt. Anschließend wird für den planaren Graphen eine planare Einbettung berechnet. Eine orthogonale Repräsentation wird dann im zweiten Schritt bestimmt. Das heißt, es werden für alle Kanten Winkel festgelegt, so dass die Anzahl der Kantenabschnitte und ihre Orientierung feststehen. Als letztes

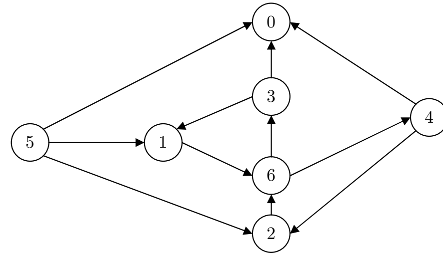
Knoten	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	1	0	0
3	1	1	0	0	0	0
4	1	0	1	0	0	0
5	1	1	1	0	0	0

(a) Graph gegeben als Adjazenzmatrix.

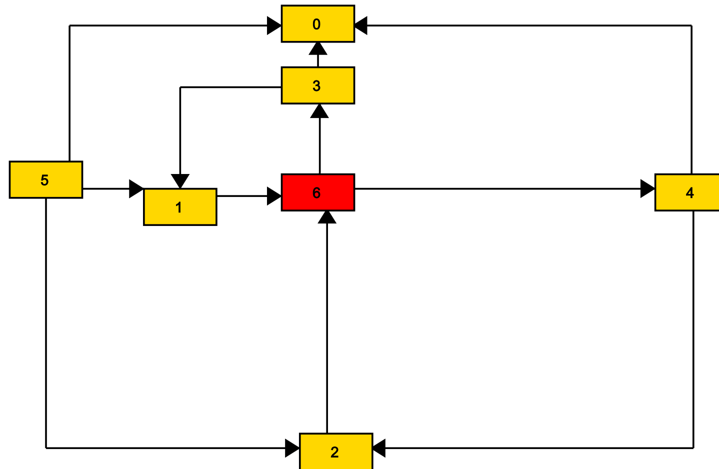
Knoten	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1
3	1	1	0	0	0	0	0
4	1	0	1	0	0	0	0
5	1	1	1	0	0	0	0
6	0	0	0	1	1	0	0

(b) Kantenkreuzungsminimierung: Einfügen eines Dummy-Knoten.

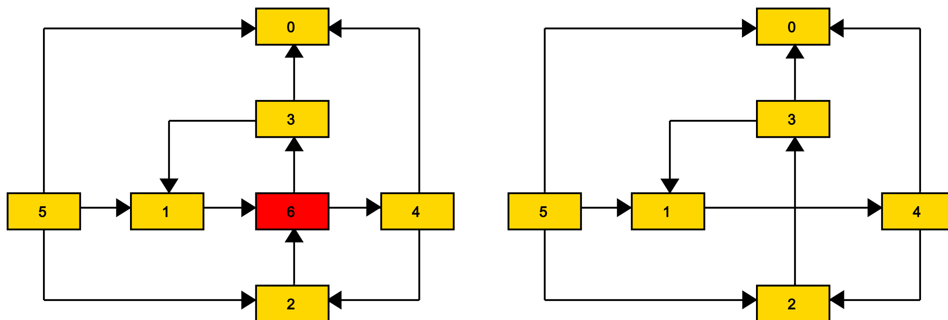
Fläche	Adjazente Kanten
f_0	$(5,0),(3,0),(3,1),(5,1)$
f_1	$(3,1),(6,3),(1,6)$
f_2	$(3,0),(4,0),(6,4),(6,3)$
f_3	$(5,1),(1,6),(2,6),(5,2)$
f_4	$(2,6),(6,4),(4,2)$
f_5 (Außenfläche)	$(5,0),(4,0),(4,2),(5,2)$



(c) Berechnung einer planaren Einbettung.



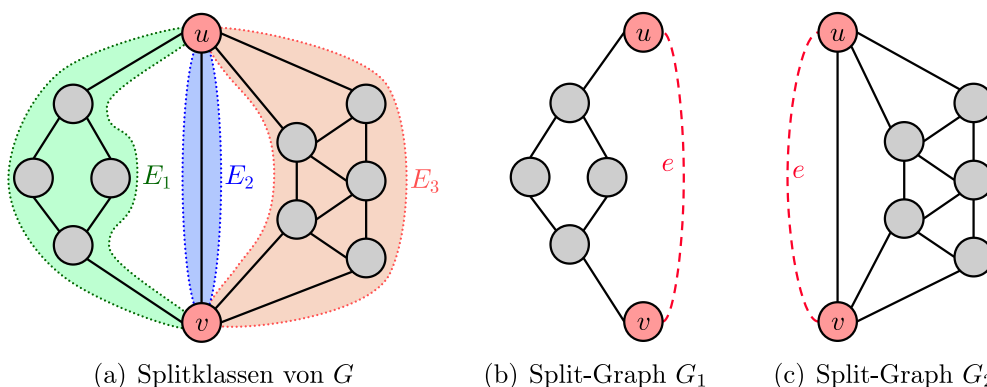
(d) Kantenwinkel werden berechnet.



(e) Länge der Kantenabschnitte wird festgelegt.

(f) Dummy-Knoten werden durch Kantenkreuzungen ersetzt.

Abbildung 2.5: Topology-Shape-Metrics Ansatz.



(a) Splitklassen von G (b) Split-Graph G_1 (c) Split-Graph G_2

Abbildung 2.6: Split-Operation für einen Graphen $G = (V, E)$ mit $C = E_1$ und $\overline{C} = E_2 \cup E_3$.

wird die Länge der Kantenabschnitte festgelegt, die x- und y-Koordinaten der Knoten bestimmt und eventuell eingefügte Dummy-Knoten durch Kantenkreuzungen ersetzt.

In Abbildung 2.5 sind die Schritte des Topology-Shape-Metrics Ansatzes an einem Beispiel veranschaulicht. Ein sehr wichtiger Schritt ist die Berechnung der planaren Einbettung. Die Güte der Zeichnung hängt entscheidend davon ab, wie gut die planare Einbettung ist, da mit ihr die grobe Gestalt der Zeichnung vorgegeben wird. Die folgenden Schritte detaillieren dann diese Gestalt zur endgültigen Zeichnung.

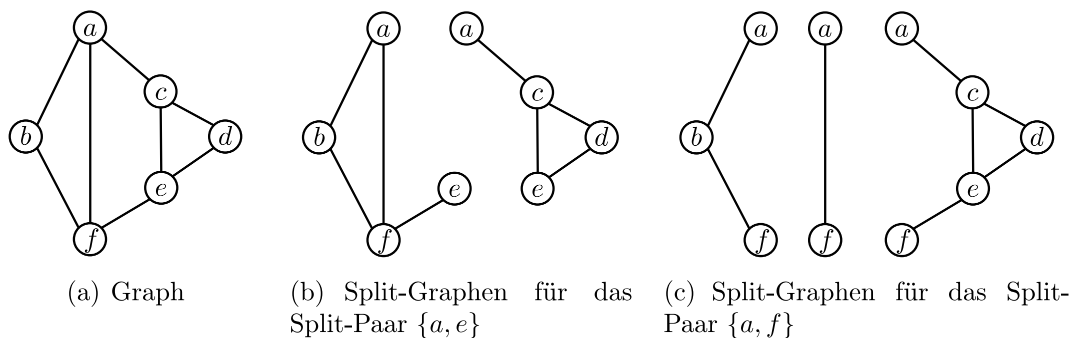
2.3 SPQR-Bäume

Für die Definition eines SPQR-Baums werden erst noch ein paar weitere Begriffe benötigt.

Definition 2.20 (Splitpaar) *Sei $G = (V, E)$ ein zweizusammenhängender Graph und $u, v \in V$, $u \neq v$. Falls $\{u, v\}$ ein Separationspaar ist oder u und v adjazente Knoten sind, heißt $\{u, v\}$ Splitpaar.*

Definition 2.21 (Splitklassen) *Sei $\{u, v\}$ ein Splitpaar. Die Splitklassen von $\{u, v\}$ sind die Elemente der Partition $E_1, \dots, E_k$ der Kantenmenge, so dass für zwei Kanten e und e' in E_i gilt, dass e und e' auf einem Weg liegen, der u und v höchstens als Endpunkt enthält (vergl. Abbildung 2.6(a)).*

Definition 2.22 (Split-Operation, Split-Graphen und virtuelle Kanten) *Seien $E_1, \dots, E_k$ die Splitklassen eines Splitpaares $\{u, v\}$. Für ein j mit $1 \leq j < k$ seien $C = E_1 \cup \dots \cup E_j$ und $\overline{C} = E_{j+1} \cup \dots \cup E_k$. Eine Split-Operation ersetzt G durch die Split-Graphen $G_1 = (V(C), C \cup e)$ und $G_2 = \left(V\left(\overline{C}\right), \overline{C} \cup e\right)$, wobei $e = \{u, v\}$ eine neue virtuelle Kante ist.*



(a) Graph

(b) Split-Graphen für das Split-Paar $\{a, e\}$

(c) Split-Graphen für das Split-Paar $\{a, f\}$

Abbildung 2.7: Das Split-Paar $\{a, e\}$ dominiert das Split-Paar $\{a, f\}$ in Bezug auf die Kante $\{d, e\}$.

Abbildung 2.6 zeigt ein Beispiel für eine Split-Operation. Ein Split-Graph eines Split-Paares $\{u, v\}$ in Bezug auf eine Kante $\{s, t\}$ in G ist die Vereinigung aller Split-Graphen von $\{u, v\}$, welche die Kante $\{s, t\}$ nicht enthalten.

Definition 2.23 (Dominieren) *Ein Split-Paar $\{u, v\}$ wird von einem anderen Split-Paar $\{x, y\}$ in Bezug auf eine Kante $\{s, t\} \in E$ dominiert, wenn der Split-Graph von $\{u, v\}$ in Bezug auf $\{s, t\}$ ein Teilgraph des Split-Graphen von $\{x, y\}$ in Bezug auf $\{s, t\}$ ist.*

In Abbildung 2.7 dominiert das Split-Paar $\{a, e\}$ das Split-Paar $\{a, f\}$ in Bezug auf die Kante $\{d, e\}$. Der Split-Graph $G_{\{a,e\}}$ von $\{a, e\}$ in Bezug auf $\{d, e\}$ enthält die Knoten $K_1 = \{a, b, e, f\}$ und der Split-Graph $G_{\{a,f\}}$ von $\{a, f\}$ in Bezug auf $\{d, e\}$ die Knoten $K_2 = \{a, b, f\}$. Somit gilt $K_2 \subset K_1$, alle Knoten aus dem Split-Graphen $G_{\{a,f\}}$ sind im Split-Graphen $G_{\{a,e\}}$ enthalten, ebenso die Kanten, und es gilt $G_{\{a,f\}}$ ist ein Teilgraph von $G_{\{a,e\}}$.

Definition 2.24 (Maximales Split-Paar) *Ein maximales Split-Paar in Bezug auf eine Kante $\{s, t\}$ in einem Graphen ist ein Split-Paar, welches durch kein anderes Split-Paar in Bezug auf $\{s, t\}$ dominiert wird.*

Es wird nun zuerst ein so genannter Proto-SPQR-Baum wie in [Wei02] definiert. Anschließend wird ein SPQR-Baum [BT96a, BT96b, DL98] definiert. Die Knoten des Proto-SPQR-Baums haben vier unterschiedliche Typen S, P, Q und R, welche in Definition 2.30 erläutert werden.

Definition 2.25 (Skelettgraph) *Jedem Knoten μ eines (Proto-)SPQR-Baums $\mathcal{T}$ ist ein Multigraph zugeordnet, welcher Skelettgraph bzw. Skelett (engl.: skeleton) genannt wird und mit* $\text{skeleton}(\mu)$ *bezeichnet wird. Ein Skelett enthält Knoten des originalen Graphen und reale, sowie virtuelle (engl.: virtual), Kanten. Reale Kanten sind Kanten des originalen Graphen. Virtuelle Kanten verbinden das Skelett mit einem anderen. Eine virtuelle Kante kann als Platzhalter für einen Teilgraphen des originalen Graphen angesehen werden. In einem Skelettgraphen* $\text{skeleton}(\mu)$ *eines $\mathcal{T}$-Knotens μ entspricht jede virtuelle Kante einer Baumkante inzident zu μ.*

Definition 2.26 (Zwillingskante) *Sei e_1 eine virtuelle Kante in* skeleton(μ) *und ν der Baumknoten, welcher e_1 entspricht. Es gibt genau eine virtuelle Kante e_2 in* skeleton(ν)*, welche μ entspricht. Man nennt e_1 die Zwillingskante von e_2 und e_2 die Zwillingskante von e_1.*

Jede virtuelle Kante in Abbildung 2.9(c) wurde in der selben Farbe wie ihre Zwillingskante gezeichnet.

Definition 2.27 (Zugehöriger Knoten) *Sei e eine virtuelle Kante in* skeleton(μ)*. Dann gibt es genau einen Knoten $\nu \in \mathcal{T}$, sodass* skeleton(ν) *die Zwillingskante von e enthält. Man bezeichnet ν als den zugehörigen Knoten (engl.: pertinent node) von e.*

Definition 2.28 (Referenzkante) *Betrachtet man zwei Knoten $\mu, \nu \in \mathcal{T}$ mit $(\mu, \nu) \in \mathcal{T}$, so gibt es eine virtuelle Kante in* skeleton(μ)*, welche eine Zwillingskante e in* skeleton(ν) *besitzt. Man nennt e dann die Referenzkante (engl.: reference edge) von ν.*

Eine alternative Definition ist die folgende: Sei $\mu \in \mathcal{T}$. Die Referenzkante von μ ist die virtuelle Kante e in skeleton(μ), so dass der zugehörige Knoten von e der Elter von μ in $\mathcal{T}$ ist. Der Skelettgraph der Wurzeln von $\mathcal{T}$ enthält keine Referenzkante.

Definition 2.29 (Pole) *Die zwei zu einer Referenzkante eines (Proto-)SPQR-Baumknotens μ inzidenten Knoten werden als Pole von μ bezeichnet.*

Der Proto-SPQR-Baum $\mathcal{T}$ eines Graphen entsteht, indem der Graph rekursiv in seine Dreizusammenhangskomponenten zerlegt wird. Die Zerlegung beginnt mit einer beliebigen Kante des Graphen, welche die Referenzkante des Proto-SPQR-Baums genannt wird. Der Knoten in $\mathcal{T}$, welcher zu dieser Kante gehört, ist die Wurzel des Proto-SPQR-Baums.

Definition 2.30 (Proto-SPQR-Baum) *Sei $G = (V, E)$ ein zweizusammenhängender Multigraph und $e = \{u, v\}$ eine Kante aus E. Sei $G' = (V, E \setminus \{e\})$ der Graph, welcher entsteht, wenn e aus G entfernt wird. Dann ist der Proto-SPQR-Baum mit Referenzkante e wie folgt definiert:*

Trivialer Fall (Q-Knoten) *G' enthält nur eine einzige Kante. In diesem Fall besteht $\mathcal{T}$ aus einem einzigen Q-Knoten. Das Skelett dieses Knotens ist G selber und besteht aus zwei Knoten und zwei Kanten.*

Serieller Fall (S-Knoten) *G' ist nicht zweizusammenhängend. Seien v_1 bis v_{k-1} mit $k > 1$ die Schnittknoten von G'. Da G zweizusammenhängend ist, hat G eine Form wie in Abbildung 2.8(a)(1) mit k Blöcken B_1 bis B_k, so dass v_i nur in den beiden Blöcken B_i und B_{i+1} für $1 \leq i \leq k-1$ enthalten ist. O.B.d.A. sei u in B_1 und v in B_k enthalten.*

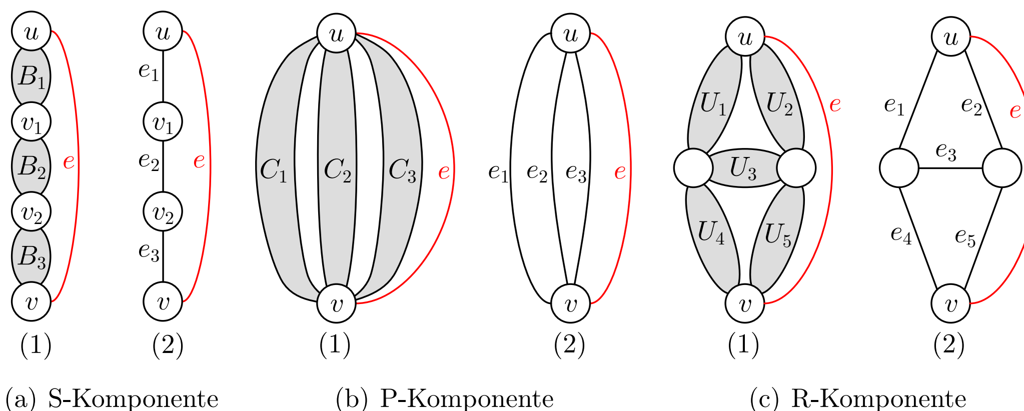


Abbildung 2.8: Struktur von zweizusammenhängenden Graphen und das Skelett der Wurzel des zugehörigen Proto-SPQR-Baums.

Die Wurzel von $\mathcal{T}$ ist ein S-Knoten μ. Das Skelett von μ ist ein Kreis, welcher die Knoten $u, v_1, \ldots, v_{k-1}, v$ in dieser Reihenfolge enthält. Sei v_0 gleich u und v_k gleich v, dann repräsentiert die Kante $e_i = (v_{i-1}, v_i)$ den Block B_i in G für $1 \leq i < k$. Das Skelett enthält diese Kanten e_i und die Kante $\{u, v\}$. Abbildung 2.8(a)(2) zeigt das Skelett des S-Knoten für den Beispielgraphen aus Abbildung 2.8(a)(1).

Die Kinder von μ werden durch die Graphen G_i definiert, welche durch Hinzufügen einer Kante $e'_i = e_i$ zum Block B_i für $1 \leq i \leq k$ konstruiert werden. Die Kante e'_i ist die Referenzkante des Skelettgraphen des i-ten Kindes von μ und gleichzeitig die Referenzkante des Proto-SPQR-Baums von G_i, wobei das i-te Kind von μ die Wurzel ist. Die Kante e'_i ist die Zwillingskante von e_i.

Paralleler Fall (P-Knoten) *Die Knoten u und v sind ein Splitpaar von G' mit den Split-Komponenten $C_1, \ldots, C_k$ mit $k \geq 2$. Abbildung 2.8(b)(1) stellt ein Beispiel für den Fall dar, dass die Wurzel des Proto-SPQR-Baums eine P-Komponente ist. Die Wurzel von $\mathcal{T}$ ist ein P-Knoten μ. Der Skelettgraph von μ besteht aus den beiden Knoten u und v und den Kanten $e_1, \ldots, e_{k+1}$. Die Kante e_i mit $1 \leq i \leq k$ stellt den Teilgraphen C_i dar, während die Kante e_{k+1} die Referenzkante von $\mathcal{T}$ ist. Abbildung 2.8(b)(2) stellt das Skelett des P-Knotens für den Graphen in Abbildung 2.8(b)(1) dar.*

Die Kinder von μ werden durch die Graphen $G_1, \ldots, G_k$ definiert, wobei G_i durch Hinzufügen einer Kante $e'_i = e_i$ zu C_i entsteht. Die Kinder von μ sind die Wurzeln der Proto-SPQR-Bäume für die Graphen G_i mit Referenzkante e'_i. Die Referenzkante des Skelettgraphen des i-ten Kindes von μ ist e'_i mit der Zwillingskante e_i.

Basisfall (R-Knoten) *Im Basisfall (engl: rigid case) trifft keine der anderen Fälle zu. G' ist also zweizusammenhängend und $\{u, v\}$ kein Splitpaar mit mindestens*

zwei Split-Komponenten. In diesem Fall ist die Wurzel von $\mathcal{T}$ ein R-Knoten μ.

Seien $\{u_1, v_1\}, \ldots, \{u_k, v_k\}$ die maximalen Split-Paare von G in Bezug auf e. Für jedes der Split-Paare $\{u_i, v_i\}$ sei U_i der Split-Graph von $\{u_i, v_i\}$ in Bezug auf e. Abbildung 2.8(c)(1) zeigt ein Beispiel für einen Graphen mit einem R-Knoten als Wurzel. Der Skelettgraph von μ beinhaltet die Knoten u und v und alle Knoten u_i und v_i für $1 \leq i \leq k$. Neben der Referenzkante e enthält skeleton(μ) *die k Kanten e_i, wobei eine Kante e_i die beiden Knoten u_i und v_i verbindet und den Teilgraphen U_i repräsentiert. Abbildung 2.8(c)(2) ist das Skelett für den Wurzel-R-Knoten des Graphen in Abbildung 2.8(c)(1).*

Die Kinder von μ werden durch die Graphen $G_1, \ldots, G_k$ definiert. G_i wird durch Hinzufügen einer Kante $e'_i = e_i$ zu U_i erzeugt. Die Kinder von μ sind die Wurzeln der Proto-SPQR-Bäume für die Graphen G_i mit Referenzkante e'_i. Die Referenzkante des Skelettgraphen des i-ten Kindes von μ ist e'_i mit der Zwillingskante e_i.

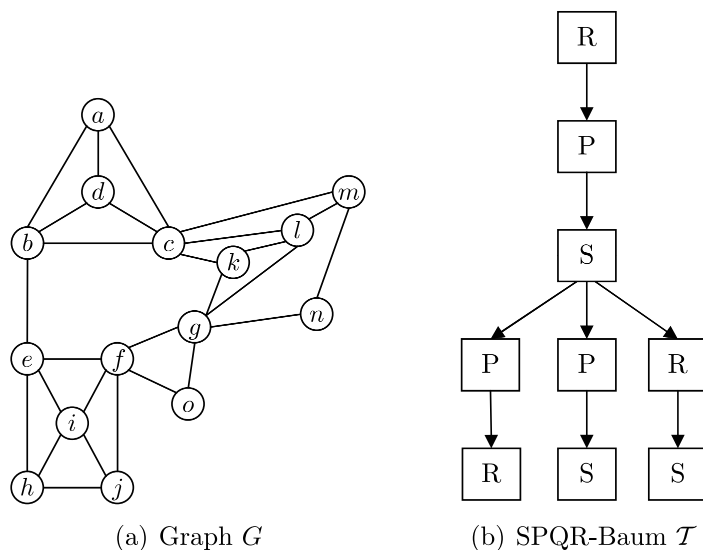
Definition 2.31 (SPQR-Baum) *Der SPQR-Baum $\mathcal{T}$ eines zweizusammenhängenden Graphen G beinhaltet einen Q-Knoten, welcher die Referenzkante e repräsentiert. Das einzige Kind dieses Knotens ist die Wurzel des Proto-SPQR-Baums für G mit Referenzkante e.*

Da für viele Anwendungen von SPQR-Bäumen, vor allem diese, welche in dieser Diplomarbeit betrachtet werden, eine Unterscheidung in S-, P- und R-Komponenten genügt, werden im folgenden die Q-Komponenten nicht betrachtet. Dies führt dazu, dass die Wurzel und die Blätter des Baumes nicht mehr Q-Knoten, sondern S-, P- oder R-Knoten sind. Würde man die Variante von SPQR-Bäumen mit Q-Knoten betrachten, würde zu jeder Kante e des originalen Graphen eine Q-Komponente gehören. Der Skelettgraph dieser Q-Komponente würde aus den beiden zu e inzidenten Knoten, e und genau einer virtuellen Kante e_v bestehen. In der Variante ohne Q-Knoten wird e direkt in den Skelettgraphen des zugehörigen Knoten von e_v an der Stelle eingefügt, wo die Zwillingskante von e_v vorhanden wäre.

In Abbildung 2.9(c) sind die Skelettgraphen der Knoten des SPQR-Baums aus Abbildung 2.9(b) abgebildet. Skelett 1 entspricht dem Wurzel-Knoten von $\mathcal{T}$. Die einzige virtuelle Kante in diesem Skelett ist die Verbindung zu Skelett 2, welches dem einzigen Kind der Wurzel in $\mathcal{T}$ entspricht.

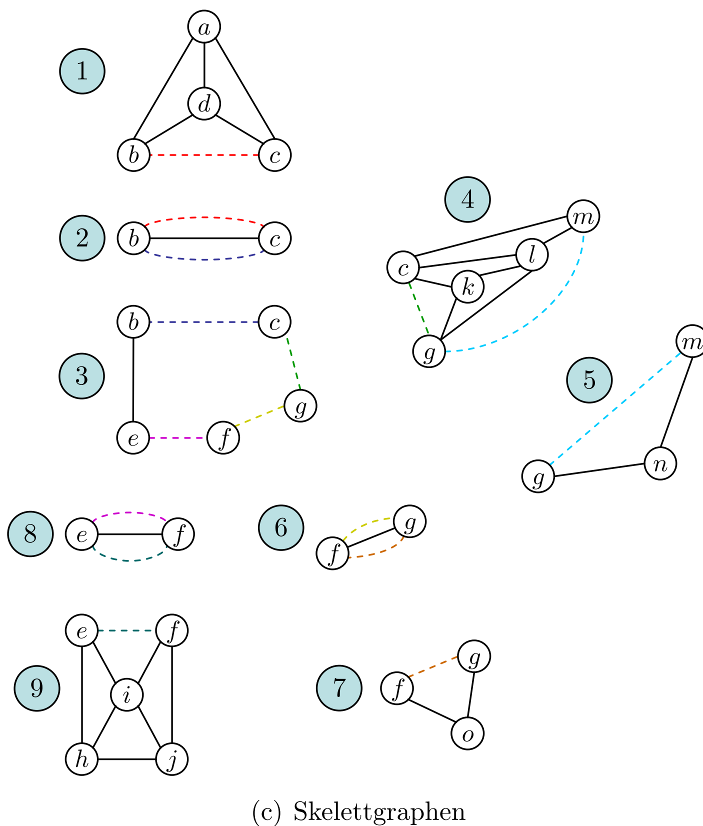
Abbildung 2.10 zeigt ein Beispiel, in dem für einen Graphen sowohl der SPQR-Baum, so wie er in der Diplomarbeit benutzt wird, berechnet wurde (Abbildung 2.10(b)), als auch der SPQR-Baum, wie er in Definition 2.31 beschrieben ist (Abbildung 2.10(c)). In dem Beispiel sind auch die Unterschiede in den Skelettgraphen zu sehen (vergl. Abbildung 2.10(e) und 2.10(f)).

Definition 2.32 (Expansionsgraph) *Sei e eine Kante in* skeleton(μ) *und ν der zugehörige Knoten von e. Das Löschen der Kante (μ, ν) würde $\mathcal{T}$ in zwei zusammenhängende Komponenten teilen. Sei $\mathcal{T}_\nu$ die Komponente, welche ν enthält. Der*



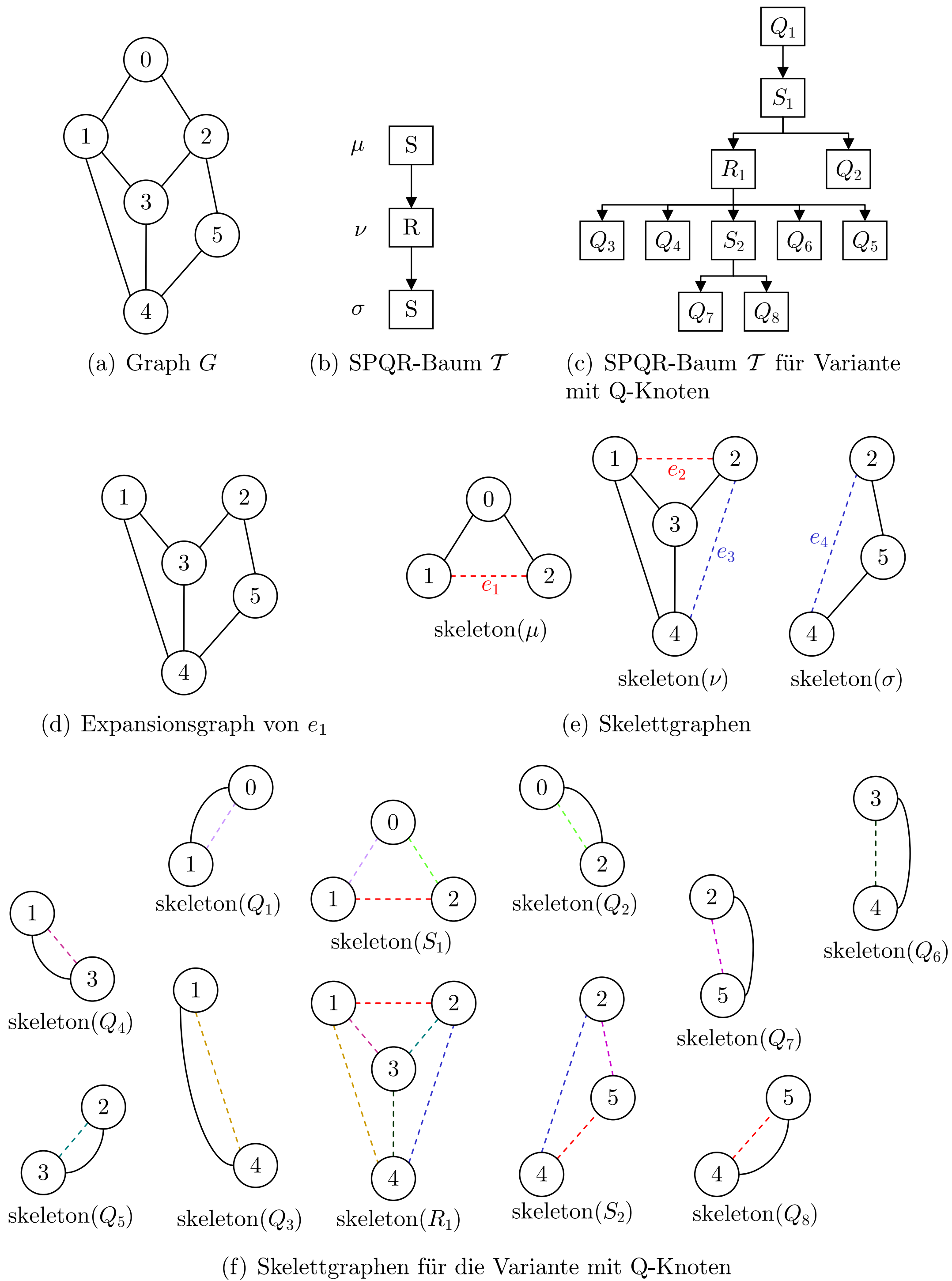
(a) Graph G

(b) SPQR-Baum $\mathcal{T}$



(c) Skelettgraphen

Abbildung 2.9: Beispiel SPQR-Baum.

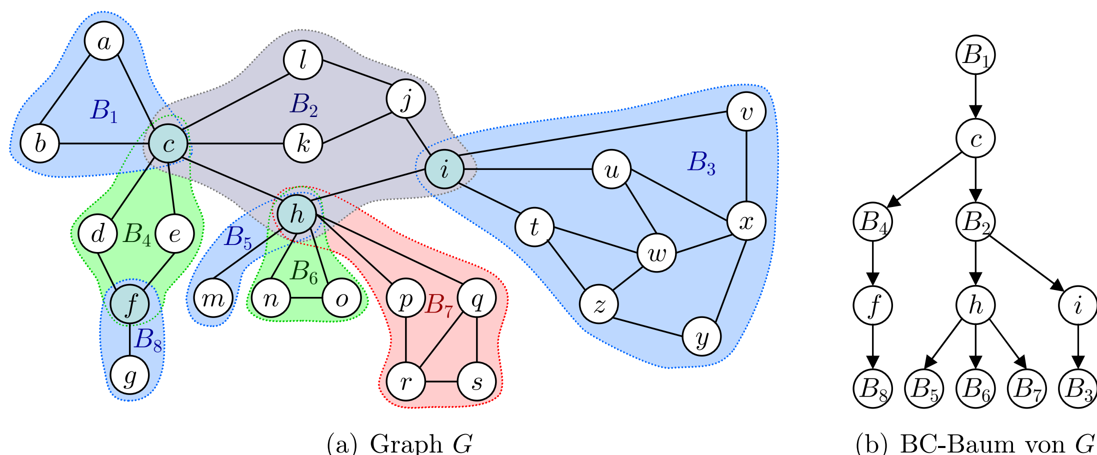


(a) Graph G (b) SPQR-Baum $\mathcal{T}$ (c) SPQR-Baum $\mathcal{T}$ für Variante mit Q-Knoten

(d) Expansionsgraph von e_1 (e) Skelettgraphen

(f) Skelettgraphen für die Variante mit Q-Knoten

Abbildung 2.10: Veranschaulichung einiger Begriffe im Zusammenhang mit SPQR-Bäumen. Der zugehörige Knoten von e_1 ist ν, μ der von e_2, σ der von e_3 und ν der von e_4. Die Referenzkante von ν ist e_2, die von σ ist e_4. Die Pole von ν sind die Knoten 1 und 2, die Pole von σ sind 2 und 4. e_2 ist die Zwillingskante von e_1 und umgekehrt, e_3 ist die Zwillingskante von e_4 und umgekehrt.



(a) Graph G (b) BC-Baum von G

Abbildung 2.11: Beispiel BC-Baum.

Expansionsgraph von e ist der Graph, welcher durch die realen Kanten in den Skelettgraphen von $\mathcal{T}_\nu$ definiert wird. Man bezeichnet ihn mit $\text{expansion}(e)$*. Weiterhin bezeichnet man mit* $\text{expansion}^+(e)$ *den Graphen* $\text{expansion}(e) \cup e$*. Das Ersetzen einer virtuellen Kante e durch ihren Expansionsgraphen nennt man expandieren von e.*

Sei μ ein Knoten in einem SPQR-Baum und e^μ_{ref} die Referenzkante von skeleton(μ). Mit „expandieren von μ" ist das Ersetzen der Zwillingskante von e^μ_{ref} durch expansion(e) gemeint. Abbildung 2.10 veranschaulicht einige der eingeführten Begriffe. SPQR-Bäume können in linearer Zeit aufgebaut werden und benötigen nur linear viel Speicherplatz [GM01].

2.4 BC-Bäume

Ein BC-Baum (engl.: block-cutvertex tree) für einen zusammenhängenden Graphen $G = (V, E)$ enthält für jeden Block von G und jeden Schnittknoten $v \in V$ einen Knoten. Zwischen einem Block-Knoten und einem Schnittknoten existiert eine Kante, falls der Schnittknoten im Block liegt. Der Wurzel-Knoten ist ein beliebiger Block-Knoten und die Kanten sind vom Elter zum Kind gerichtet. Abbildung 2.11 zeigt ein Beispiel für einen BC-Baum eines Graphen. BC-Bäume können durch einen modifizierten Tiefensuche-Algorithmus [Tar72] in linearer Laufzeit mit linear viel Speicherplatz aufgebaut werden.

Definition 2.33 (Erweiterter dualer BC-Baum) *Bei der Definition des erweiterten dualen BC-Baums (engl.: extended dual BC-tree) einer planaren Einbettung* (Γ, f) *werden zwei Fälle unterschieden:*

1. *Alle Kanten der Außenfläche gehören zu demselben Block B in G:*

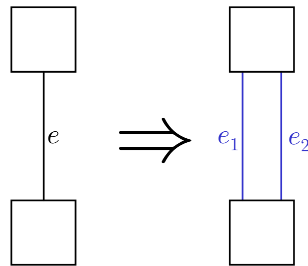


Abbildung 2.12: Ersetzen der Kante e durch die Multikanten e_1 und e_2.

Der erweiterte duale BC-Baum ist dann der BC-Baum des dualen Graphen von G mit Wurzel b. Zusätzlich wird ein Knoten r und eine Kante (r, b) eingefügt. Dabei ist b der Block-Knoten, welcher mit B assoziiert ist.

2. *Die Kanten der Außenfläche gehören nicht zu demselben Block in G:*

   In diesem Fall ist der erweiterte duale BC-Baum der BC-Baum des dualen Graphen von G mit Wurzel r, wobei r der Schnittknoten zugehörig zu f in G ist.

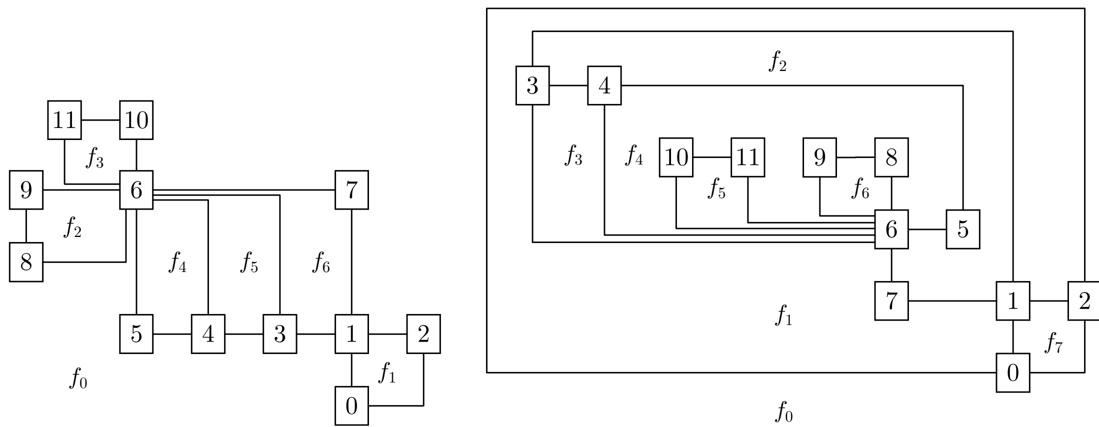
Abbildung 2.13(e) und Abbildung 2.13(f) zeigen Beispiele für erweiterte duale BC-Bäume. Mit Hilfe von BC-Bäumen und dualen Graphen kann die Tiefe einer planaren Einbettung eines Graphen definiert werden.

Definition 2.34 (Tiefe einer planaren Einbettung) *Für eine planare Einbettung (Γ, f) ist die Tiefe (engl.: depth) definiert als die Tiefe des erweiterten dualen BC-Baums.*

In Abbildung 2.13 ist ein Beispiel gegeben, in dem die Tiefen von zwei planaren Einbettungen berechnet werden.

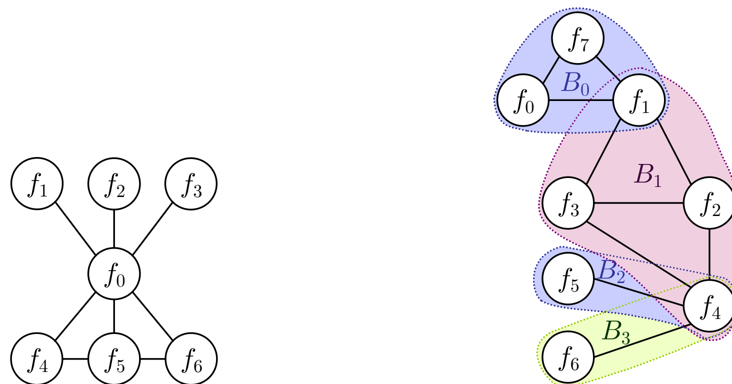
Im Rahmen der Diplomarbeit wird der Begriff der erweiterten Tiefe benötigt, der an dieser Stelle eingeführt werden soll. Die Tiefe einer planaren Einbettung zeigt den Zusammenhang zwischen Blöcken und Flächen. Wenn ein Block in einer Fläche liegt, so handelt es sich dabei um eine Schnittfläche. Bei der Definition werden allerdings nur Blöcke betrachtet, welche mindestens eine eigene Fläche besitzen. Dagegen werden so genannte simple Blöcke, welche aus genau zwei Knoten und einer Kante bestehen, nicht in die Berechnung der Tiefe einbezogen. Im Unterschied dazu werden bei der Berechnung der erweiterten Tiefe einer planaren Einbettung alle Blöcke mit einbezogen. Bevor die erweiterte Tiefe formal definiert wird, wird zunächst ein erweiterter dualer Graph und anschließend ein erweiterter Block-Cutface-Baum definiert.

Definition 2.35 (Erweiterter dualer Graph) *Sei $G = (V, E)$ ein Graph und Γ eine kombinatorische Einbettung von G. Der Graph $G' = (V, E')$ und die Einbettung Γ' entstehen, indem in allen Blöcken in G, die genau eine Kante $e \in E$ enthalten, e durch die Multikanten $e_1, e_2 \in E'$ ersetzt wird (vergl. Abbildung 2.12). Der erweiterte duale Graph von Γ ist der duale Graph von Γ'.*



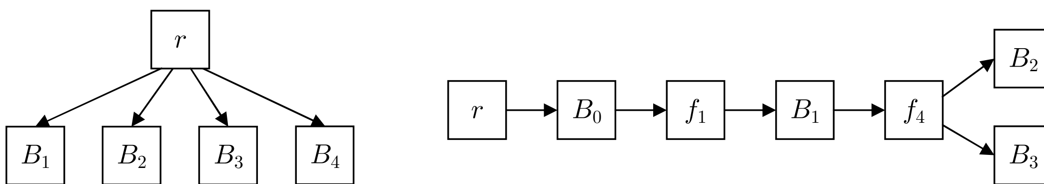
(a) Graph G_1 (Quelle: [PT00])

(b) Graph G_2 (Quelle: [PT00])



(c) Dualer Graph von G_1

(d) Dualer Graph von G_2



(e) Erweiterter dualer BC-Baum von $G_1 \Rightarrow$ Tiefe 1

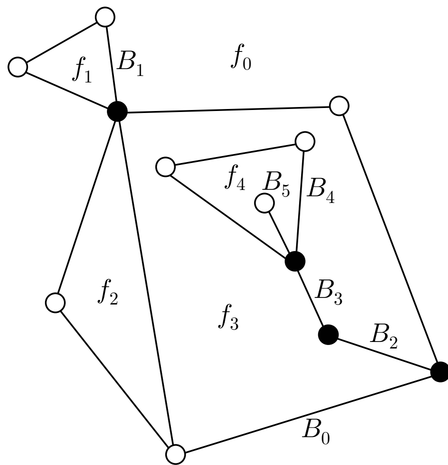
(f) Erweiterter dualer BC-Baum von $G_2 \Rightarrow$ Tiefe 5

Abbildung 2.13: Berechnung der Tiefe eines Graphen. G_1 und G_2 sind zwei verschiedene Einbettungen desselben Graphen.

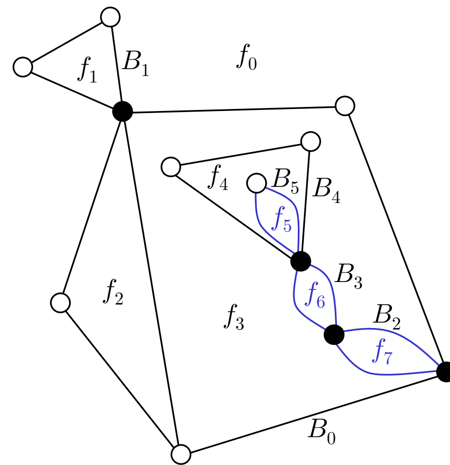
Definition 2.36 (Erweiterter Block-Cutface-Baum) *Der erweiterte Block-Cutface-Baum einer planaren Einbettung* (Γ, f) *ist der erweiterte duale BC-Baum der Einbettung, wobei anstatt des dualen Graphen der erweiterte duale Graph benutzt wird.*

Definition 2.37 (Erweiterte Tiefe einer planaren Einbettung) *Die erweiterte Tiefe einer planaren Einbettung* (Γ, f) *ist die Tiefe des erweiterten Block-Cutface-Baums von* (Γ, f)*.*

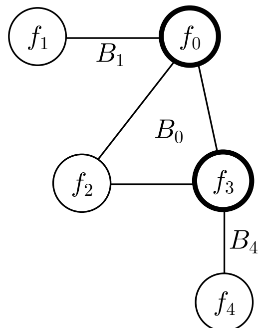
Abbildung 2.14 veranschaulicht an einem Beispiel den Unterschied zwischen den verschiedenen Tiefen-Definitionen.



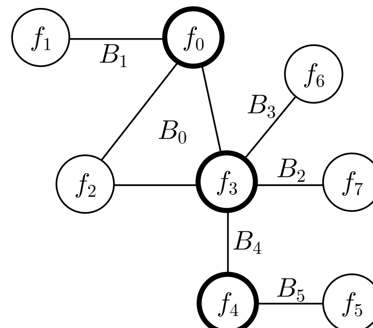
(a) Graph $G = (V, E)$ mit planarer Einbettung (f_0, Γ)



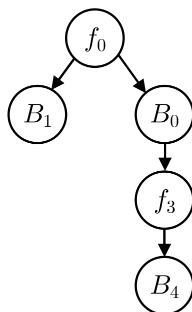
(b) Graph $G' = (V, E')$ mit planarer Einbettung (f_0, Γ')



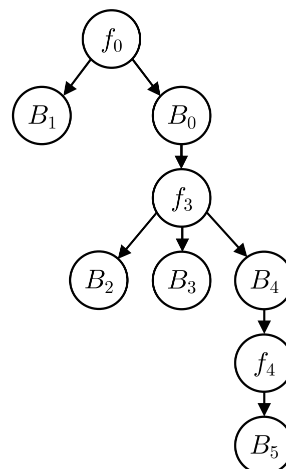
(c) dualer Graph



(d) erweiterter dualer Graph



(e) erweiterter dualer BC-Baum



(f) erweiterter Block-Cutface-Baum

Abbildung 2.14: Vergleich Tiefe einer Einbettung (hier 3) mit erweiterter Tiefe einer Einbettung (hier 5).

3 Berechnung einer planaren Einbettung für planare Graphen

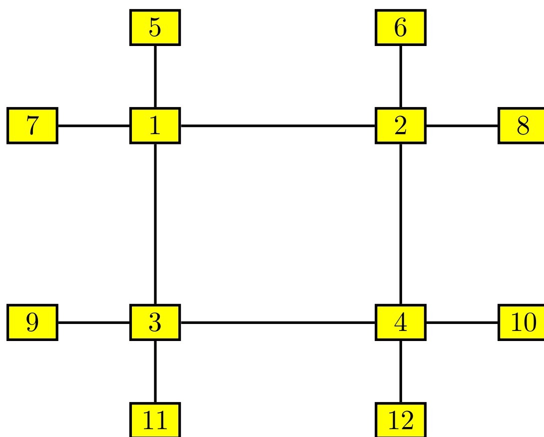
In diesem Kapitel werden einige Algorithmen vorgestellt, die für einen planaren Graphen eine planare Einbettung berechnen. Diese Algorithmen können im Topology-Shape-Metrics Ansatz im ersten Schritt benutzt werden, nachdem der Graph planarisiert wurde (vergl. Kapitel 2.2).

Eine Zeichnung zu finden, welche minimale Kantenknickanzahl hat, wenn die planare Einbettung des Graphen nicht fixiert ist, ist NP-schwer [GT01]. Es existieren zwar Algorithmen, die das Problem lösen, jedoch sind sie aufgrund der exponentiellen Worst-Case Rechenzeit nur für Graphen mit bis zu 80 Kanten effizient [MW02]. Wie viele verschiedene Einbettungen es geben kann wird deutlich, wenn man Abbildung 3.1 betrachtet. Der Graph besteht aus 12 Knoten und 12 Kanten. Alleine durch Verändern des Eintrags in der Adjazenzmatrix für Knoten 1 können 6 verschiedene kombinatorische Einbettungen generiert werden: $\{5, 2, 3, 7\}$, $\{5, 2, 7, 3\}$, $\{2, 3, 5, 7\}$, $\{2, 5, 3, 7\}$, $\{2, 5, 7, 3\}$ und $\{2, 7, 5, 3\}$. Insgesamt gibt es für diesen Graphen 1296 unterschiedliche kombinatorische Einbettungen.

Es werden daher Optimierungskriterien für planare Einbettungen und Algorithmen, welche eine planare Einbettung mit optimalem Wert für diese Kriterien berechnen, betrachtet. Der Algorithmus von Pizzonia und Tamassia [PT00] in Kapitel 3.1 hat als Optimierungsparameter die Tiefe der planaren Einbettung. In [Piz05] wurde experimentell gezeigt, dass planare Einbettungen mit minimaler Tiefe gegenüber Einbettungen mit großer Tiefe bessere Ergebnisse liefern bezüglich Bewertungskriterien wie der Kantenknickanzahl oder der Summe der Kantenlängen der Zeichnungen. Die Ergebnisse sind in Kapitel 3.1.1 zusammengefasst.

Die Kapitel 3.2, 3.3 und 3.4 beschreiben die Ergebnisse des Papers von Gutwenger und Mutzel [GM04b]. In dem Paper wird ein Algorithmus vorgestellt, der ebenfalls die Tiefe der planaren Einbettung minimiert. Im Gegensatz zum Algorithmus von Pizzonia und Tamassia funktioniert er aber auch für Graphen, in denen die Einbettung der Zweizusammenhangskomponenten nicht vorgegeben und fix ist. Der im Paper beschriebene Algorithmus berechnet eine Einbettung mit minimaler erweiterter Tiefe (vergl. Definition 2.37), er kann jedoch leicht modifiziert werden, so dass er eine Einbettung mit minimaler Tiefe berechnet. Diese Modifikation wird in Kapitel 3.3.1 beschrieben. Es werden noch zwei weitere Algorithmen vorgestellt. Ein Algorithmus maximiert die Außenfläche des Graphen und ein Algorithmus berechnet über alle Einbettungen mit minimaler Tiefe eine Einbettung mit einer Außenfläche maximaler Größe.

Im folgenden wird angenommen, dass die planaren Graphen keine Multikanten

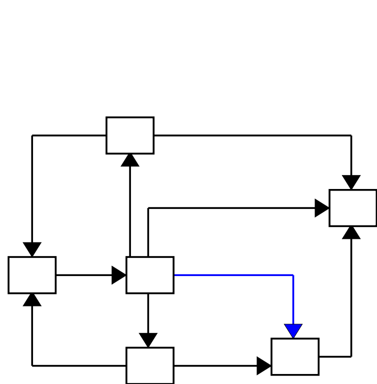


(a) Zeichnung

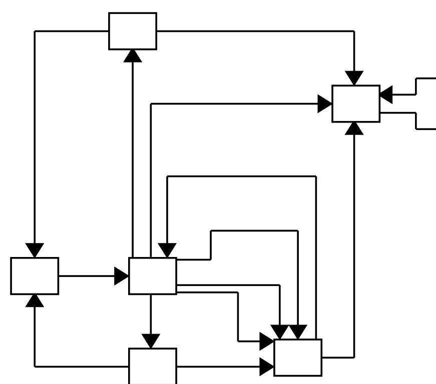
Knoten	inzidente Kanten
1	5, 2, 3, 7
2	6, 8, 4, 1
3	1, 4, 11, 9
4	2, 10, 12, 3
5	1
6	2
7	1
8	2
9	3
10	4
11	3
12	4

(b) Adjazenzmatrix (inzidente Kanten sind durch zweiten zur Kante inzidenten Knoten angegeben)

Abbildung 3.1: Beispiel für die Anzahl verschiedener Einbettungen.



(a) berechnete planare Einbettung



(b) wieder eingefügte Multikanten und Self Loops

Abbildung 3.2: Multikanten und Self Loops.

und Self Loops enthalten. Der Ausschluss von Multikanten ist keine Einschränkung, da diese zu einer einzigen Kante zusammengefasst werden können. Nach der Berechnung der planaren Einbettung kann diese Kante dann wieder durch die originalen Kanten ersetzt werden. Es wäre zwar auch denkbar, die Kanten an verschiedene Stellen einzubetten, für die Übersichtlichkeit wäre dies aber schlechter, als sie alle nebeneinander an derselben Stelle zu platzieren. Ebenso können Kanten von einem Knoten auf sich selber entfernt werden und nach dem Berechnen der planaren Einbettung wieder an irgendeiner Stelle der Adjazenzliste des Knotens eingefügt werden. Die Stelle, an der diese Kante eingefügt wird, beeinflusst die Kantenknickanzahl nicht. Die Größe der Flächen beispielsweise wird jedoch wohl beeinflusst, daher wäre die Wahl der Position in der Adjazenzliste für dieses Kriterium nicht unerheblich. Die Wahl soll jedoch nicht von den Algorithmen behandelt werden. In Abbildung 3.2 ist ein Beispiel dargestellt, in dem ein Graph Multikanten und einen Self Loop enthält. Zuerst wurde eine planare Einbettung für einen Graphen, in dem die Multikanten zu einer Kante (blau dargestellt) zusammengefasst wurden, berechnet. Anschließend wurden die Multikanten und der Self Loop wieder eingefügt.

3.1 Minimale Tiefe Algorithmus von Pizzonia und Tamassia

Pizzonia und Tamassia stellen in [PT00] einen Algorithmus vor, der eine planare Einbettung für einen allgemeinen Graphen berechnet, welche minimale Tiefe hat (vergl. Definition 2.34). Ihr Algorithmus beschränkt sich dabei allerdings auf Graphen, in denen die Einbettung der zweizusammenhängenden Komponenten vorgegeben und fix ist. Dieses Kapitel beschreibt, wie der Algorithmus funktioniert. Für Details wie Korrektheitsbeweise sei auf das Paper [PT00] verwiesen.

Als erstes wird eine Lösung vorgestellt, die das folgende Problem löst. Sie wird für die Lösung des allgemeinen Falls benötigt.

Problem 3.1.1 (Eingeschränkte Tiefen-Minimierung) *Gegeben ist ein planarer Graph G, eine kombinatorische Einbettung für jeden Block in G und ein Schnittknoten v in G. Gesucht ist eine planare Einbettung Γ_G^* für G, welche die Einbettungen der Blöcke nicht verändert, so dass v in der Außenfläche liegt und Γ_G^* minimale Tiefe über alle möglichen planaren Einbettungen Γ_G hat.*

Algorithmus 3.1 löst Problem 3.1.1. Sei $\mathcal{B}$ der BC-Baum von G, c ein Schnittknoten in G und B ein Block in G mit einer gegebenen kombinatorischen Einbettung Γ_B, welcher c enthält. Die Menge der möglichen Schnittflächen von (B, c) ist die Menge der Flächen in Γ_B, welche c enthalten.

Sei b ein Knoten, ein Schnittknoten oder Block, in $\mathcal{B}$. Sei $\mathcal{B}_b$ der zusammenhängende Teilbaum von $\mathcal{B}$, welcher entsteht, wenn aus $\mathcal{B}$ die Kante vom Elter von b zu b entfernt wird, und welcher b enthält. Der Algorithmus berechnet rekursiv für alle Knoten b in $\mathcal{B}$ eine planare Einbettung $\Gamma(b)$ für den Graphen, welcher durch die

Schnittknoten und Blöcke in $\mathcal{B}_b$ definiert ist. Die Tiefe einer dieser planaren Einbettungen $\Gamma(v)$ bzw. $\Gamma(B)$ ist nach den Gleichungen 3.1 und 3.4 definiert. Dabei sei f_B die Außenfläche von $\Gamma(B)$.

$$\begin{aligned} \mathrm{Tiefe}(\Gamma(B)) &:= \max\{d_P, d_{NP}\} && (3.1)\\ d_P &:= \max_{v \in f_B}\big(\mathrm{Tiefe}(\Gamma(v))\big) && (3.2)\\ d_{NP} &:= 2 + \max_{v \notin f_B}\big(\mathrm{Tiefe}(\Gamma(v))\big) && (3.3)\\ \mathrm{Tiefe}(\Gamma(v)) &:= \max_{B \in \mathrm{Kinder}(v)}\big(\mathrm{Tiefe}(\Gamma(B))\big) && (3.4) \end{aligned}$$

Algorithmus 3.1 : Eingeschränkte Tiefen-Minimierung

$\Gamma_G :=$ `betteEin`(v)
return Γ_G

Funktion `betteEin`(*Schnittknoten v*)

$\Gamma(v) := \emptyset$
forall $(v, B) \in \mathcal{B}$ **do**
  if B ist ein Blatt in $\mathcal{B}$ **then**
    $\Gamma(B) := \Gamma_B$ mit irgendeiner möglichen Schnittfläche von (B, v) als Außenfläche
  else
    $\Gamma(B) :=$ `betteEin`(B)
  bette $\Gamma(B)$ in die Außenfläche von $\Gamma(v)$ ein
end
return $\Gamma(v)$

Beginnend bei Schnittknoten v, welcher in der Außenfläche liegen muss, berechnet der Algorithmus für alle Blöcke adjazent zu v eine Einbettung und bettet diese in die Außenfläche ein. Beim Berechnen der Einbettung eines Blocks B wird als erstes für alle Schnittknoten in B ungleich dem Elter rekursiv eine Einbettung berechnet. Für alle möglichen Schnittfächen f von B und dem Elter von B wird der Wert $\delta(f)$ berechnet. Dieser ist die maximale Tiefe der Einbettung $\Gamma(v')$ über alle Schnittknoten v' in der Fläche f. Die mögliche Schnittfläche f_B mit maximalem δ-Wert wird als Schnittfläche gewählt. Existieren mehrere Flächen mit maximalem δ-Wert, wird die Fläche gewählt, welche die maximale Anzahl Schnittknoten v' mit maximalem Tiefe($\Gamma(v')$)-Wert enthält. Alle Einbettungen $\Gamma(v')$ der Schnittknoten v' in B werden, falls möglich, in f_B eingebettet. Falls dies nicht möglich ist, kann die Einbettung in eine beliebige Fläche eingebettet werden.

Algorithmus 3.1 wird nun dazu benutzt, den allgemeinen Fall der Tiefe-Minimierung zu lösen (Problem 3.1.2).

Funktion `betteEin`(*Block* B)

forall $(B, v) \in \mathcal{B}$ **do**
  $\Gamma(v) :=$ `betteEin`(v)
end
$\Theta :=$ alle möglichen Schnittflächen von $(B, \text{Elter}(B))$
forall $f \in \Theta$ **do**
  $\delta(f) := \max\limits_{v \in \mathcal{B}, v \in f} \big(\text{Tiefe}(\Gamma(v))\big)$
end
$f_B := f \in \Theta$ mit $\delta(f) = \max_{f \in \Theta} \delta(f)$ und maximaler Anzahl Schnittknoten v mit tiefster Einbettung $\Gamma(v)$
forall $(B, v) \in \mathcal{B}$ **do**
  if $v \in f_B$ **then**
    ordne Schnittfläche f_B dem Schnittpaar (B, v) zu
  else
    ordne dem Schnittpaar (B, v) eine beliebige Schnittfläche zu
end
$\Gamma(B) :=$ Füge zur gegebenen Einbettung $\Gamma(B)$ die berechneten planaren Einbettungen $\Gamma(v)$ der Kinder von B hinzu, wobei für ein Kind v von B die Einbettung $\Gamma(v)$ in die Schnittfläche eingefügt wird, welche (B, v) zugeordnet wurde.
return $\Gamma(B)$

Problem 3.1.2 (Tiefen-Minimierung) *Gegeben sei ein planarer zusammenhängender Graph G und eine kombinatorische Einbettung für jeden Block B in G. Gesucht ist eine planare Einbettung Γ_G mit minimaler Tiefe, welche die Einbettungen der Blöcke nicht verändert.*

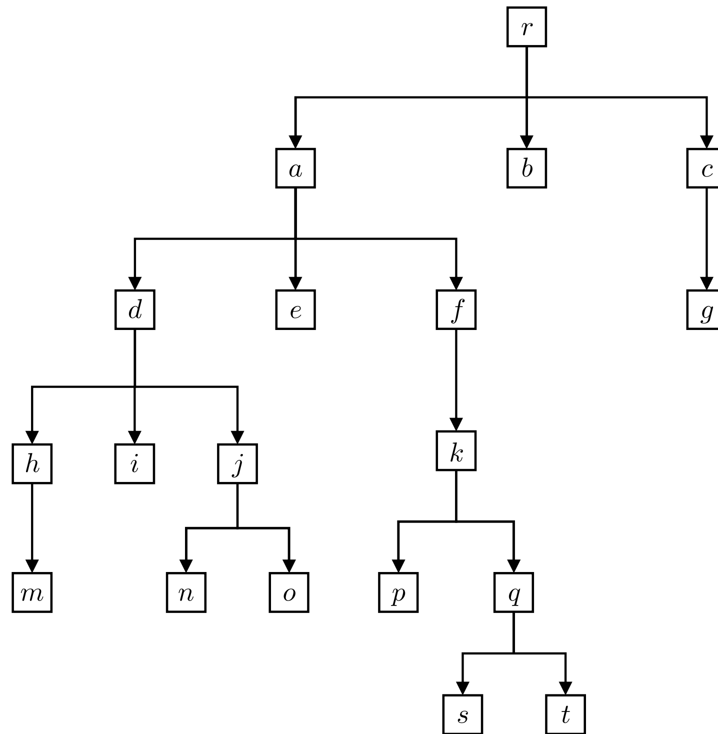
Die Distanz zwischen zwei Knoten eines Baums sei die Länge eines kürzesten Weges zwischen den Knoten. Der Durchmesser eines Baumes ist die maximale Distanz zwischen zwei Blättern des Baums. Der Durchmesser einer planaren Einbettung ist der Durchmesser des dualen BC-Baums. Als Durchmesser-Pfad wird ein Weg zwischen zwei Blättern mit maximaler Länge bezeichnet. Der Durchmesser-Baum ist die Vereinigung aller Durchmesser-Pfade. Sei T ein Baum mit Wurzel r und T_{diam} sein Durchmesser-Baum. Der Knoten von T_{diam}, welcher die geringste Distanz zu r hat, wird mit $\kappa(T)$ bezeichnet. Abbildung 3.3 veranschaulicht diese Begriffe.

Die Tiefe eines Baums T ist die maximale Distanz der Wurzel zu einem Blatt. Ein Tiefenpfad ist ein Pfad von der Wurzel zu einem Blatt mit maximaler Distanz zur Wurzel. Der Tiefenbaum T_{depth} ist die Vereinigung aller Tiefenpfade von T.

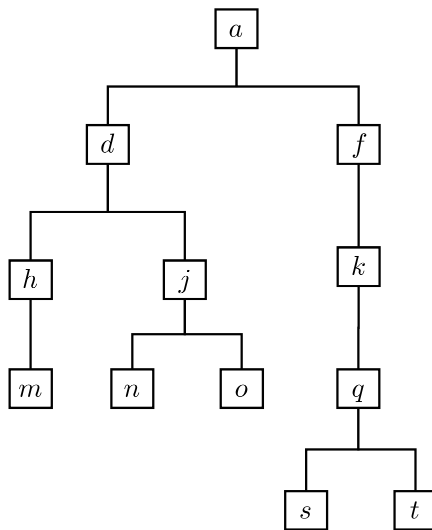
Der Algorithmus, um Problem 3.1.2 zu lösen, besteht aus den folgenden Schritten:

1. Wähle einen beliebigen Schnittknoten v und benutze Algorithmus 3.1, um eine Einbettung Γ_G mit minimaler Tiefe und v auf der Außenfläche zu berechnen.

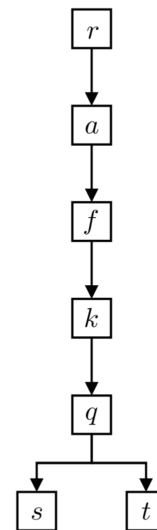
2. Finde κ des dualen BC-Baums von Γ_G und, wenn nötig, modifiziere Γ_G zu einer Einbettung mit minimalem Durchmesser mittels Algorithmus 3.4.



(a) Baum T



(b) Durchmesser-Baum T_{diam}



(c) Tiefenbaum T_{depth}

Abbildung 3.3: Ein Baum, sein Durchmesser- und Tiefenbaum. Der Durchmesser von T ist 7 und die Durchmesser-Pfade sind $m \to s$, $m \to t$, $n \to s$, $n \to t$, $o \to s$ und $o \to t$. Der Knoten a ist der Knoten aus T_{diam}, welcher die kürzeste Distanz zu r in T hat, somit gilt $\kappa(T) = a$. Die Tiefenpfade sind $r \to s$ und $r \to t$.

Algorithmus 3.4 : Modifiziere Γ zu einer Einbettung mit minimalem Durchmesser

```
k := κ(dualer BC-Baum)
if k ist ein Block then
    if k ist nicht Wurzel des Tiefenbaums T_depth des dualen BC-Baums then
        T_diam := Durchmesser-Baum des dualen BC-Baums
        forall Schnittknoten w, (k, w) ∈ T_diam do
            Ξ(w) := {Fläche f | f ist mögliche Schnittfläche von (k, w)}
        end
        if ∃ eine Fläche f : ∀(k, w) ∈ T_diam : f ∈ Ξ(w) then
            ordne allen Schnittknoten w mit (k, w) ∈ T_diam die Fläche f zu
        end
    end
end
```

3. Wähle eine Außenfläche, welche die Tiefe der planaren Einbettung minimiert. Diese Fläche ist ein Schnittflächen-Knoten im dualen BC-Baum von Γ_G mit minimaler Distanz zu einem Blatt.

Die Laufzeit und der Speicherplatzbedarf des Algorithmus sind linear.

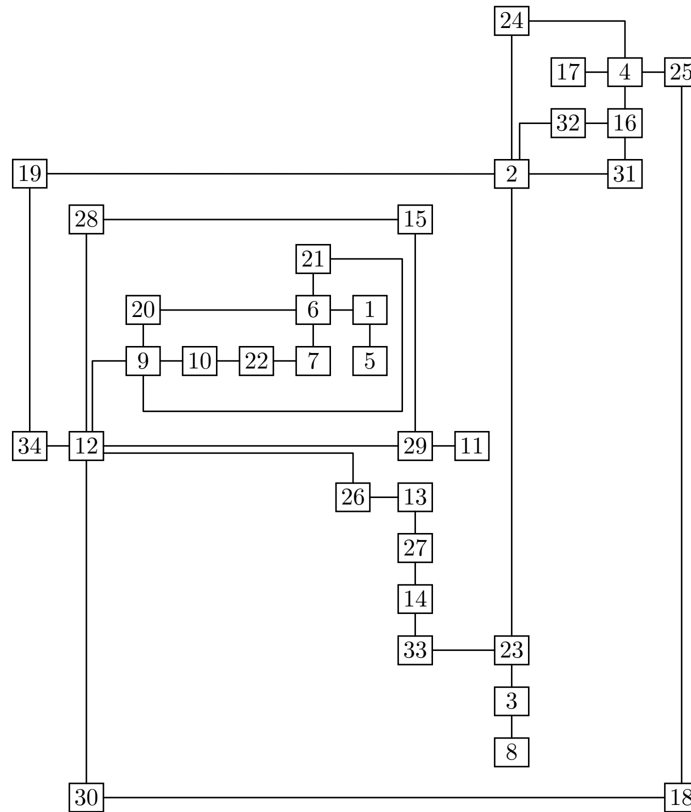
3.1.1 Experimentelle Analyse

Pizzonia hat in [Piz05] den vorgestellten Algorithmus für minimale Tiefe mit einem Algorithmus verglichen, welcher eine planare Einbettung mit großer Tiefe berechnet. Wie dieser Algorithmus für große Tiefe funktioniert, wird in dem Paper [Piz05] beschrieben und ist für den Vergleich unerheblich.

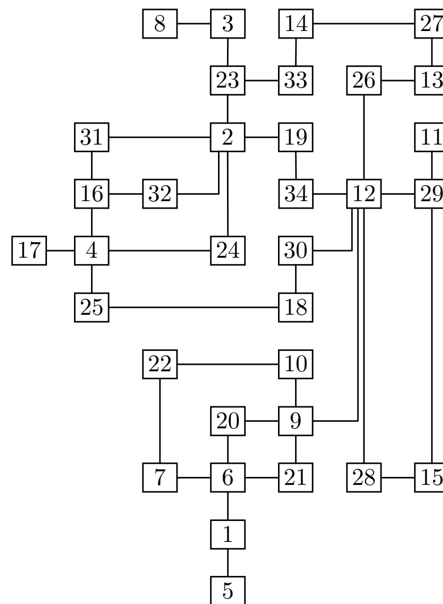
Die Abbildungen 3.5 und 3.4 zeigen 2 Beispielgraphen. Beide Graphen wurden jeweils einmal mit dem Algorithmus für minimale Tiefe von Pizzonia und Tamassia und einmal mit dem in dem Paper vorgestellten Algorithmus gezeichnet, welcher eine Einbettung mit großer Tiefe berechnet.

Die Experimente wurden mit zwei Sammlungen von Testgraphen durchgeführt. Die erste Sammlung beinhaltet Graphen, welche von reale Welt Graphen abgeleitet wurden. Dazu wurden Graphen aus Anwendungen, wie z.B. Datenbanken, genommen und diese Graphen durch Hinzufügen von Knoten und Kanten variiert, um so aus wenigen Graphen eine größere Menge Testgraphen zu erzeugen (Details in [BG+97]). Die Testgraphensammlung kann auf der GDToolkit Webseite [GDT] runtergeladen werden.

Die zweite Sammlung besteht aus 410 randomisiert erzeugten Graphen. Diese Graphen sind planar und haben eine kleine Trivialität und kleine maximale Belegung. Die Trivialität eines Graphen ist das Verhältnis zwischen der Anzahl von Blöcken mit genau einer Kante und der Anzahl der Kanten des Graphen. Sie gibt die Ähnlichkeit des Graphen zu einem Baum wieder. Ein Baum hat Trivialität von

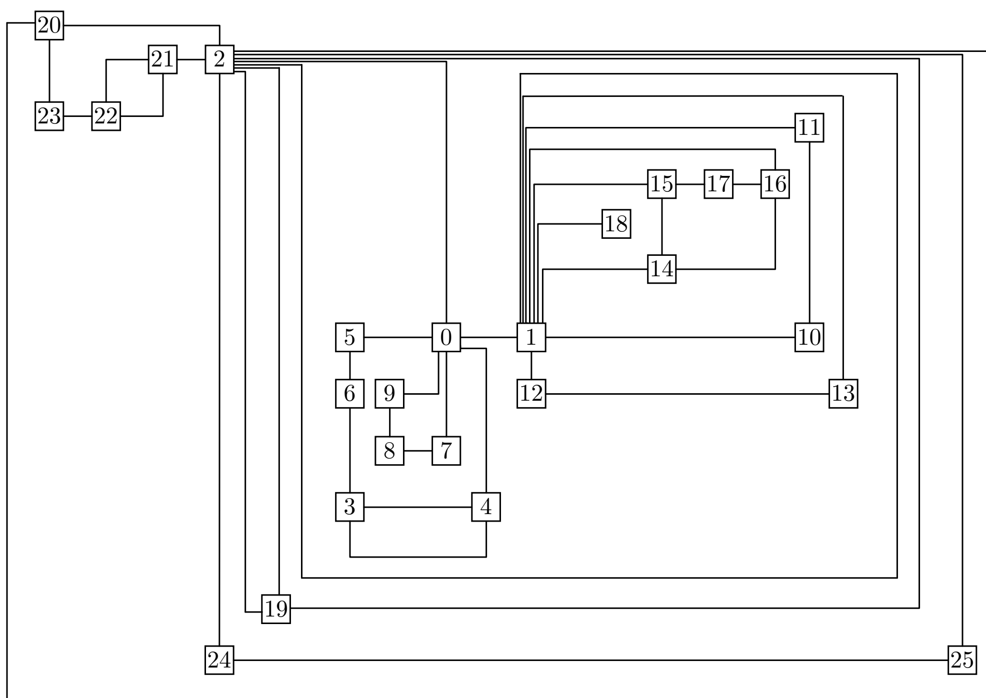


(a) Graph erstellt mit Algorithmus für große Tiefe.

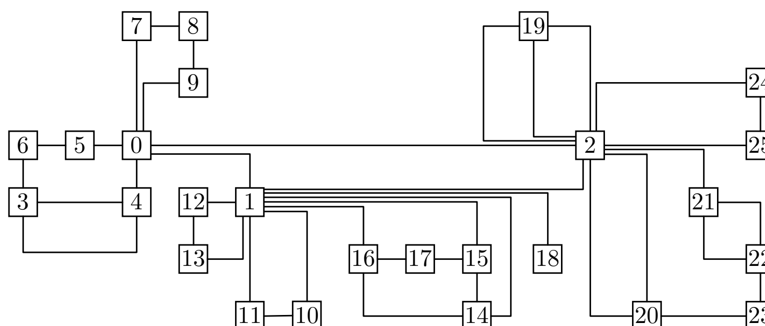


(b) Graph erstellt mit Algorithmus für minimale Tiefe.

Abbildung 3.4: Vergleich von zwei Einbettungen mit großer und minimaler Tiefe für einen reale Welt Graphen. (Quelle: [Piz05])



(a) Graph erstellt mit Algorithmus für große Tiefe.



(b) Graph erstellt mit Algorithmus für minimale Tiefe.

Abbildung 3.5: Vergleich von zwei Einbettungen mit großer und minimaler Tiefe für einen randomisiert erstellen Graphen. (Quelle: [Piz05])

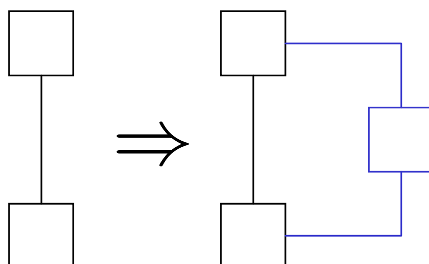


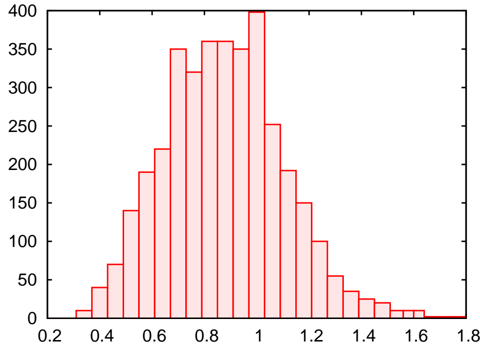
Abbildung 3.6: Einfügen eines Dummy-Knotens mit zwei Dummy-Kanten.

1, ein zweizusammenhängender Graph hingegen eine Trivialität von 0. Die maximale Belegung eines Graphen ist die maximale Belegung eines Blocks des Graphen. Die Belegung eines Blocks ist das Verhältnis zwischen der Anzahl der Kanten des Blocks und der Gesamtanzahl der Kanten des Graphen. Die maximale Belegung gibt die Ähnlichkeit eines Graphen zu einem zweizusammenhängenden Graphen wieder. Ein zweizusammenhängender Graph hat eine maximale Belegung von 1, ein Baum eine maximale Belegung von beinahe 0. Die Anzahl der Knoten n der Graphen liegt zwischen 10 und 50, die Anzahl der Schnittknoten zwischen $n/10$ und $n/5$ und jeder Schnittknoten ist zu 2 bis 5 Blöcken adjazent. Kein Block ist zu mehr als 5 Schnittknoten adjazent. In [Piz01] sind Details für den Erzeugungsalgorithmus gegeben.

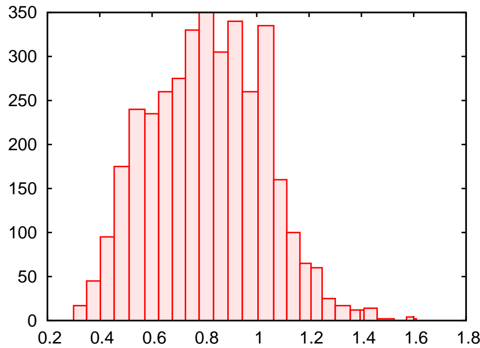
Untersucht wurde der Einfluss der Tiefe der Einbettung auf die Größe der benötigten Zeichenfläche (Area), die Summe der Kantenlängen (EdgeLength) und die Anzahl der Kantenknicke (Bends). Die Abbildungen 3.7 und 3.8 zeigen die Ergebnisse. Dabei bezeichnet Γ_{min} das Ergebnis des Algorithmus für minimale Tiefe und Γ_{large} das Ergebnis des Algorithmus für großer Tiefe. Die y-Achse stellt jeweils die Anzahl der Zeichnungen dar, die den x-Wert besitzen. Es ist zu erkennen, dass bei randomisiert erzeugten Graphen der Vorteil der minimalen Tiefe groß ist. Bei den reale Welt Graphen ist zwar auch ein Vorteil erkennbar, dieser ist allerdings nicht so groß wie bei den randomisiert erzeugten Graphen.

3.1.2 Modifikation für minimale erweiterte Tiefe

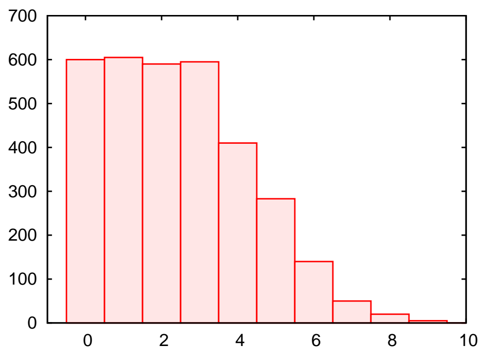
Der Algorithmus lässt sich leicht modifizieren, so dass er eine Einbettung mit minimaler erweiterter Tiefe berechnet. Dazu wird vor dem Aufruf des Algorithmus in jedem Block, welcher genau eine Kante besitzt, ein Dummy-Knoten und jeweils eine Dummy-Kante zu den beiden Knoten des Blocks eingefügt (vergl. Abbildung 3.6). Nach der Berechnung der Einbettung werden diese Dummy-Knoten und Kanten wieder entfernt. Durch das Einfügen der Dummy-Knoten und Kanten wird eine Fläche erzeugt. Somit hat der Block durch diese Fläche einen Einfluss auf den dualen Graphen und den dualen BC-Baum.



(a) $\mathrm{Area}(\Gamma_{\mathrm{min}})/\mathrm{Area}(\Gamma_{\mathrm{large}})$

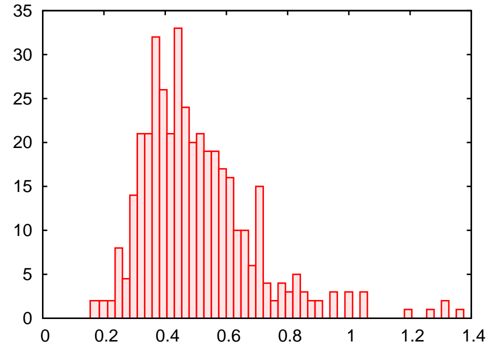


(b) $\mathrm{EdgeLength}(\Gamma_{\mathrm{min}})/\mathrm{EdgeLength}(\Gamma_{\mathrm{large}})$

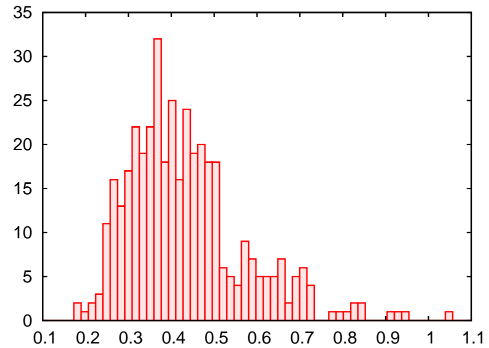


(c) $\mathrm{Bends}(\Gamma_{\mathrm{large}}) - \mathrm{Bends}(\Gamma_{\mathrm{min}})$

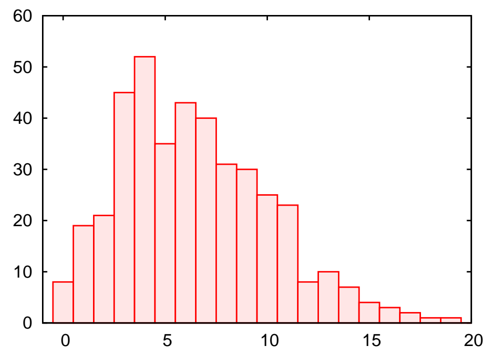
Abbildung 3.7: Messungen für reale Welt Graphen. (Quelle: [Piz05])



(a) $\mathrm{Area}(\Gamma_{\mathrm{min}})/\mathrm{Area}(\Gamma_{\mathrm{large}})$



(b) $\mathrm{EdgeLength}(\Gamma_{\mathrm{min}})/\mathrm{EdgeLength}(\Gamma_{\mathrm{large}})$



(c) $\mathrm{Bends}(\Gamma_{\mathrm{large}}) - \mathrm{Bends}(\Gamma_{\mathrm{min}})$

Abbildung 3.8: Messungen für randomisiert erzeugte Graphen. (Quelle: [Piz05])

3.2 Maximale Außenfläche

Die Algorithmen in diesem und den folgenden Kapiteln stammen aus dem Paper [GM04b]. Es wird hier zusammengefasst, wie die Algorithmen funktionieren. Die Beweise der Korrektheit und Laufzeit werden jedoch weggelassen, für sie sei auf das Paper verwiesen.

Gesucht ist eine planare Einbettung für einen planaren Graphen $G = (V, E)$, in der die Größe der Außenfläche maximal ist. In Kapitel 3.2.1 wird als erstes ein Algorithmus für das Finden einer Einbettung mit maximaler Außenfläche und die Berechnung der Größe der maximalen Außenfläche, welche einen gegebenen Knoten v enthält, für zweizusammenhängende Graphen vorgestellt. Kapitel 3.2.2 beschäftigt sich dann mit der Verallgemeinerung auf zusammenhängende Graphen.

3.2.1 Zweizusammenhängende Graphen

Sei $B = (V_B, E_B)$ ein Block in G und $\mathcal{T}_B$ sein SPQR-Baum. Jedem Knoten und jeder Kante wird eine Länge zugeordnet. Die Länge $\ell(e)$ einer Kante $e \in E_B$ ist gegeben und fix. Die Länge einer virtuellen Kante e wird auf die Komponentenlänge (siehe Definition 3.1) von e gesetzt. Die Knotenlängen $\ell(v)$ für alle Knoten $v \in V$ sind ebenfalls gegeben und fix. Die Größe size(f) einer Fläche f ist wie folgt definiert:

$$\text{size}(f) = \sum_{e \in f} \ell(e) + \sum_{v \in f} \ell(v) \tag{3.5}$$

Definition 3.1 (Komponentenlänge) *Sei $e = \{v, w\}$ eine Kante im Skelettgraphen und sei Γ_e eine Einbettung des Expansionsgraphen* expansion$^+(e)$*, so dass Γ_e eine Fläche f^* enthält, welche e enthält und maximale Größe über alle Einbettungen des Expansionsgraphen hat.*

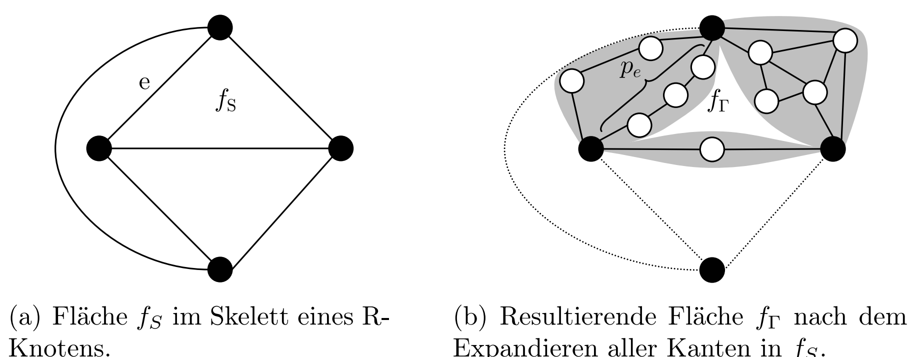
Die Komponentenlänge (engl.: component length) ist definiert als:

$$\text{complength}(e) = \text{size}(f^*) - \ell(e) - \ell(v) - \ell(w) \tag{3.6}$$

Bestimmung der Größe einer maximalen Außenfläche

Die Grundidee des Algorithmus ist es, Flächen zu expandieren. Um Fläche f_S zur Fläche f_Γ zu expandieren, müssen alle virtuellen Kanten, die zur Fläche f_S gehören, durch Einbettungen der zugehörigen Skelette ersetzt werden. Das bedeutet, dass alle virtuellen Kanten e durch ihren Expansionsgraphen expansion$^+(e)$ ersetzt werden. Dabei wird die Kante e im Skelettgraphen durch einen Weg p_e im Expansionsgraphen ersetzt. Die Fläche f_Γ wird so groß wie möglich gemacht, indem die Länge der Wege p_e maximiert wird (vergl. Abbildung 3.9). Diese Idee wird in Lemma 3.1 festgehalten.

Lemma 3.1 *Sei B ein planarer, zweizusammenhängender Graph mit Kantenlängen und $\mathcal{T}_B$ sein SPQR-Baum. Es gilt:*



(a) Fläche f_S im Skelett eines R-Knotens.

(b) Resultierende Fläche f_Γ nach dem Expandieren aller Kanten in f_S.

Abbildung 3.9: Expandieren von Fläche f_S zur Fläche f_Γ.

(i) Jede Fläche f im Skelett kann expandiert werden zu einer Fläche f', so dass die Größe der Fläche $\text{size}(f')$ *der Summe der Längen der Knoten in f plus der Komponentenlängen aller Kanten in f entspricht.*

(ii) Es existiert ein innerer Knoten $\mu \in \mathcal{T}_B$ und eine Einbettung Γ_μ von $\text{skeleton}(\mu)$, *so dass eine Fläche f in Γ_μ existiert, welche zu einer maximalen Fläche f^* von B expandiert werden kann. Alle Kanten in f werden expandiert zu Expansionsgraphen mit maximaler Länge.*

Die Komponentenlängen werden wie folgt bestimmt: Zunächst wird die Komponentenlänge für alle virtuellen Kanten, welche keine Referenzkante sind, in einem bottom-up-Traversal von $\mathcal{T}_B$ berechnet. Sei e eine virtuelle Kante in $\text{skeleton}(\mu)$ und ν der zugehörige Knoten von e. Sei e_r die Referenzkante von ν und L die Summe der Längen der beiden Pole von μ. Die Länge aller Kanten in $\text{skeleton}(\nu)$, mit Ausnahme der Referenzkante e_r, sei gleich der Komponentenlänge. Dann wird $\text{complength}(e)$ abhängig von der Art des Knotens ν wie folgt definiert:

- S-Knoten: Größe einer beliebigen der zwei Flächen in $\text{skeleton}(\nu)$ minus L
- P-Knoten: Länge der längsten Kante ungleich e_r in $\text{skeleton}(\nu)$
- R-Knoten: Größe der größten Fläche im Skelettgraphen von ν, welche e_r enthält, minus L

Nun können die Komponentenlängen der Referenzkanten berechnet werden. Dazu wird ein top-down-Traversal von $\mathcal{T}_B$ benutzt. Sei $\mu \in \mathcal{T}_B$ und S das Skelett von μ. Mit $e_{S,\nu}$ wird die Zwillingskante der Referenzkante von ν in μ bezeichnet. Die Komponentenlänge aller Kanten in S ist bekannt, da der Skelettgraph der Wurzel keine Referenzkante enthält und die Komponentenlänge der Referenzkanten aller Kinder vor dem rekursiven Aufruf berechnet wurde. Sei ν ein Kind von μ. Unterschieden wird der Knotentyp von μ:

- S-Knoten: Sei L die Summe der Längen aller Knoten und Kanten in S. Die Komponentenlänge der Referenzkante von ν ist L minus der Länge von $e_{S,\nu}$ minus der Länge der beiden Knoten inzident zu $e_{S,\nu}$.

- P-Knoten: Die Komponentenlänge der Referenzkante von ν ist die Länge der längsten Kante in S ungleich $e_{S,\nu}$.
- R-Knoten: Sei f die größte Fläche in S, welche $e_{S,\nu}$ enthält. Die Komponentenlänge der Referenzkante von ν ist size(f) minus der Länge von $e_{S,\nu}$ minus der Längen der beiden Knoten inzident zu $e_{S,\nu}$.

Nach Lemma 3.1 muss eine Einbettung Γ_μ für ein Skelett mit einer Fläche f von maximaler Größe über alle möglichen Einbettungen gefunden werden. Dazu werden alle Skelette S betrachtet:

- Wenn S das Skelett eines R-Knotens ist, existieren genau zwei spiegelsymmetrische Einbettungen. Es wird dann die größte Fläche in irgendeiner dieser Einbettungen gewählt.
- Handelt es sich um einen P-Knoten der parallelen Kanten $e_1, \ldots, e_k$, so werden die zwei längsten Kanten e_i und e_j, $i \neq j$, gewählt, welche die größte Fläche erzeugen.
- Im Falle eines S-Knotens gibt es nur eine einzige Einbettung für S, welche aus zwei identisch großen Flächen besteht.

Es kann also ein Baumknoten μ und eine Skelettfläche $f \in \Gamma_\mu$ bestimmt werden, welche zu einer maximalen Fläche von B expandiert werden kann. Nach Lemma 3.1 müssen alle Kanten in f zu Expansionsgraphen mit maximaler Länge expandiert werden, was durch rekursives Aufrufen des Algorithmus erreicht werden kann. Der Algorithmus berechnet in linearer Zeit eine planare Einbettung von B mit maximaler Außenfläche.

Sei $\mathcal{M}$ die Menge der Knoten μ im SPQR-Baum $\mathcal{T}_B$, dessen Skelettgraphen v enthalten. Um die Größe der maximalen Außenfläche zu berechnen, welche den Knoten v enthält, müssen alle Knoten μ in $\mathcal{M}$ betrachtet werden. Wenn die Komponentenlänge aller virtuellen Kanten berechnet wurde, kann in dem Skelettgraph von μ die Größe einer maximalen Fläche berechnet werden. Die Größe der maximalen Außenfläche, welche v enthält, ist die Größe einer maximalen Fläche, welche v enthält, in den Skelettgraphen der Knoten $\mu \in \mathcal{M}$. Auch dieser Algorithmus benötigt lineare Laufzeit.

Berechnung einer planaren Einbettung mit maximaler Außenfläche

Um eine planare Einbettung (Γ, f_{ext}) für B mit maximaler Außenfläche zu berechnen, werden zunächst die Kantenlängen für alle virtuellen Kanten berechnet. Dann wird, wie zuvor beschrieben, ein Knoten μ^* bestimmt, dessen Skelettgraph eine Fläche maximaler Größe enthält. O.B.d.A. enthält skeleton(μ^*) mindestens eine reale Kante. Die Idee ist die folgende: Für skeleton(μ^*) wird eine planare Einbettung mit maximaler Außenfläche berechnet. Dann wird diese Einbettung zu Γ hinzugefügt, indem die Kanten aller Knoten in skeleton(μ^*) in der Reihenfolge, die die planare

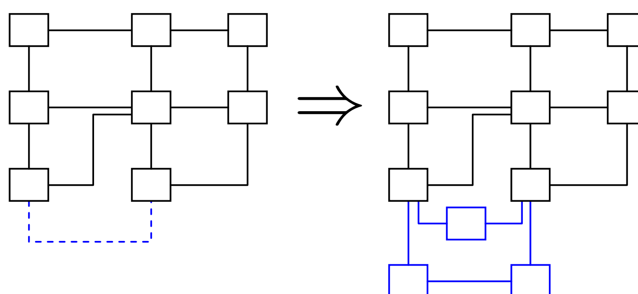
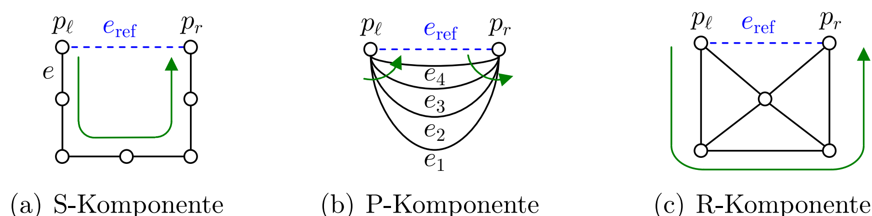


Abbildung 3.10: Ersetzen einer virtuellen Kante durch ihren Expansionsgraphen.



(a) S-Komponente (b) P-Komponente (c) R-Komponente

Abbildung 3.11: Darstellung der Skelettgraphen mit fester Position der Referenzkante, p_ℓ und p_r.

Einbettung vorgibt, zu Γ hinzugefügt werden. Falls eine Kante virtuell ist, wird sie durch ihren Expansionsgraphen ersetzt und dieser an die Stelle der virtuellen Kante eingefügt. Dazu wird beim rekursiven Aufruf des Algorithmus gespeichert, an welche Stelle der beiden zur virtuellen Kante inzidenten Knoten der Expansionsgraph eingefügt werden soll (vergl. Abbildung 3.10). Die Idee wird im folgenden Algorithmus umgesetzt.

Gesucht ist für jeden Knoten in B die Reihenfolge der inzidenten Kanten und die Außenfläche f_{ext}. Für jeden Knoten v in B wird eine Liste $A(v)$ mit der Reihenfolge der inzidenten Kanten berechnet. Zu Beginn ist $A(v)$ für alle Knoten leer. Weiterhin wird für die Pole jeder Referenzkante ein Wert $b(v)$ gespeichert. Dieser ist zu Beginn für alle Pole nicht definiert und wird vor dem rekursiven Aufruf der Einbettungsfunktion für einen Knoten $\mu \in \mathcal{T}$ auf den Eintrag in $A(v)$ gesetzt, vor den die Kanten des Skelettgraphen von μ eingefügt werden sollen. Falls der Wert nicht definiert ist, werden die Elemente an das Ende der Liste angefügt. Vor dem rekursiven Aufruf wird ebenfalls festgelegt, welcher der beiden Pole der Referenzkante von skeleton(μ) der linke Knoten ist. Dabei wird angenommen, dass die Skelettgraphen so dargestellt werden wie in Abbildung 3.11. Die Referenzkante wird immer ganz oben gezeichnet und der linke Pol p_ℓ befindet sich auf der linken Seite.

Die Einbettungsfunktion `betteEin`(μ) unterscheidet anhand der Art von μ. Sei e^μ_{ref} die Referenzkante von skeleton(μ) und p^μ_ℓ der linke Pol, p^μ_r der rechte. Im folgenden wird nur bestimmt, welcher der beiden Pole p^μ_ℓ ist, da sich damit automatisch ergibt, dass der andere p^μ_r ist. In einer Einbettung sei $\text{next}_v(e)$ in der Reihenfolge der zu v inzidenten Kanten der Eintrag nach der Kante e und $\text{prev}_v(e)$ der Eintrag vor

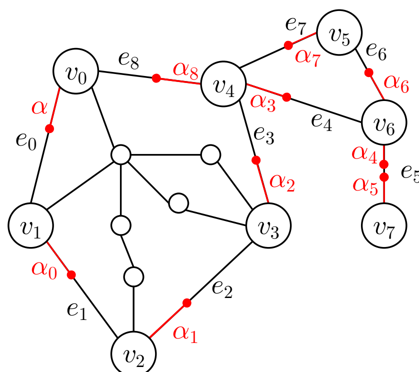


Abbildung 3.12: Berechnen der Außenfläche bei gegebenem Adjazenzeintrag α. Die Außenfläche ist $f_{\text{ext}} = \{v_0, e_0, v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, v_6, e_5, v_7, e_5, v_6, e_6, v_5, e_7, v_4, e_8\}$.

Funktion `FügeKanteHinzu(`*Knoten v, Kante e, Eintrag a in A(v)*`)`

---

if e ist reale Kante **then**
    Füge e vor a in $A(v)$ ein, a' sei diese Stelle.
else
    $\nu :=$ zugehöriger Knoten von e
    if ν wurde noch nicht betrachtet **then**
        if zugehöriger Knoten von v in skeleton$(\nu) = p^\nu_\ell$ **then**
            $b(p^\nu_\ell) := a$
        else
            $b(p^\nu_r) := a$
        `betteEin`(ν)
    end
    if $e = e^\mu_{\text{ref}}$ **then**
        if $v = p^\mu_\ell$ **then**
            $a' := b(p^\mu_\ell)$
            $b(p^\mu_\ell) := a$
        else
            $a' := b(p^\mu_r)$
            $b(p^\mu_r) := a$
        end
    else
        if zugehöriger Knoten von v in skeleton$(\nu) = p^\nu_\ell$ **then**
            $a' := b(p^\nu_\ell)$
        else
            $a' := b(p^\nu_r)$
end
return a'

---

e. Sei $\text{twin}_v(e)$ für eine zu v inzidente Kante e der zu e inzidente Knoten ungleich v. Durch den Adjazenzeintrag $\alpha = (v, e)$ wird die Außenfläche definiert. α wird im Algorithmus gesetzt und definiert f_{ext} rekursiv wie folgt. Der Knoten $v \in \alpha$ und die Kante $e \in \alpha$ liegen in der Außenfläche. Sei $\alpha_0 = (v_0, e_0)$ mit $v_0 = \text{twin}_v(e)$ und $e_0 = \text{prev}_{v_0}(e)$. Sowohl v_0 als auch e_0 sind ebenfalls in f_{ext}. Genauso $\alpha_1 = (v_1, e_1)$ mit dem Knoten $v_1 = \text{twin}_{v_0}(e_0)$ und der Kante $e_1 = \text{prev}_{v_1}(e_0)$ usw. In dieser Definition wird α wieder erreicht, nachdem alle Knoten und Kanten in f_{ext} bestimmt wurden. Abbildung 3.12 veranschaulicht diese Berechnung für einen zusammenhängenden Graphen. Es ist offensichtlich, dass sie auch für zweizusammenhängende Graphen funktioniert.

S-Knoten Die Kanten des Skelettgraphen werden in der seriellen Reihenfolge, beginnend bei p_ℓ, zu Γ hinzugefügt (vergl. grüner Pfeil in Abbildung 3.11(a)). Dabei wird die Einbettungsfunktion rekursiv für virtuelle Kanten aufgerufen. Dies geschieht automatisch in der Funktion `FügeKanteHinzu`, welche für eine reale oder virtuelle Kante aufgerufen wird. Entweder die reale Kante wird direkt eingebettet oder ein Rekursionsaufruf gestartet.

Falls μ nicht die Wurzel von $\mathcal{T}$ ist, sei $v = p_\ell^\mu$, e die zu p_ℓ^μ inzidenten Kante ungleich e_{ref}^μ und $a = b(p_\ell^\mu)$. Falls μ die Wurzel von $\mathcal{T}$ ist, wird eine beliebige reale Kante e gewählt, v sei ein beliebiger zu e inzidenter Knoten, a sei nicht definiert und es wird α gleich (v, e) gesetzt. Es wird nun die Funktion `SKnotenSchleife`(v, e, a) aufgerufen.

Funktion `SKnotenSchleife`(*Knoten v, Kante e, Eintrag a in A(v)*)

```
if e ist virtuell then
    ν := zugehöriger Knoten von e
    p_ℓ^ν := zugehöriger Knoten von v in skeleton(ν)
    if twin_v(e) = p_r^μ then
        b(p_r^ν) := b(p_r^μ)
end
a' := FügeKanteHinzu(v, e, a)
if v = p_ℓ^μ then
    b(p_ℓ^μ) := a'
v' := twin_v(e)
if v = p_r^μ then
    a'' := b(p_r^ν)
a''' := FügeKanteHinzu(v', e, a'')
if μ ist nicht Wurzel von T oder v ∉ α then
    if v ≠ p_r^μ then
        SKnotenSchleife(v', next_v'(e), a''')
    else
        b(p_r^μ) := a'''
end
```

P-Knoten Die Kanten in skeleton(μ) seien $e_1, \ldots, e_k$. O.B.d.A. sei e_1 die längste Kante in skeleton(μ) ungleich e^μ_{ref} und e_k gleich e^μ_{ref}, falls μ nicht die Wurzel von $\mathcal{T}$ ist. Der Wert a wird zu Beginn gleich $b(p^\mu_\ell)$ gesetzt. Die Funktion `EinbettungVonPKomponente` wird zuerst für die Parameter p^μ_ℓ, e_i und a aufgerufen für $i = 1, \ldots, k$. Zuletzt wird $b(p^\mu_\ell)$ gleich a gesetzt.

Nun müssen die Kanten noch einmal für p^μ_r betrachtet werden, und zwar in umgekehrter Reihenfolge. a wird zu Beginn gleich $b(p^\mu_r)$ gesetzt. Die Funktion `EinbettungVonPKomponente` wird für die Parameter p^μ_r, e_i und a aufgerufen für $i = k, \ldots, 1$. Dann wird $b(p^\mu_r)$ gleich a gesetzt.

Falls μ der Wurzelknoten von $\mathcal{T}$ ist, existiert eine reale Kante e_{vref}, welche als alternative Referenzkante betrachtet wird. Das heißt, diese Kante wird an die Stelle von e_{ref} in Abbildung 3.11(b) eingebettet, ein beliebiger Knoten $p_{\text{v}\ell}$ als alternativer linker Pol betrachtet und α gleich $(p_{\text{v}\ell}, e_{\text{vref}})$ gesetzt.

Funktion `EinbettungVonPKomponente(`*Knoten v, Kante e, Eintrag a in A(v)*`)`

$u := p^\mu_\ell$, falls $v = p^\mu_r$, sonst $u := p^\mu_r$
if e ist virtuell **then**
  $\nu :=$ der zugehörige Knoten von e
  $p^\nu_\ell := p^\mu_\ell$
  $u' :=$ der zu u zugehörige Knoten in skeleton(ν)
  $b(u') := b(u)$
end
$a :=$ `FügeKanteHinzu`(v, e, a)

R-Knoten Es existieren genau zwei spiegelsymmetrische Einbettungen für skeleton(μ). Zunächst wird eine beliebige Einbettung gewählt und eine maximale Fläche f^μ_{ext} berechnet, welche e^μ_{ref} enthält. Falls die Vorgängerkante der Referenzkante in der Adjazenzliste von p^μ_ℓ nicht in f^μ_{ext} ist, muss die Einbettung gespiegelt werden, damit sie zu dem Schema in Abbildung 3.11(c) passt.

Zuerst wird die Funktion `EinbettungVonRKomponente` für alle Knoten in f^μ_{ext} aufgerufen, in der Reihenfolge wie sie in Abbildung 3.11(c) durch den grünen Pfeil dargestellt ist. Für die virtuellen Kanten der Außenfläche ist dabei wichtig, dass der linke Pol so gewählt wird, dass die virtuelle Kante in Fläche f^μ_{ext} expandiert wird. Anschließend wird `EinbettungVonRKomponente` für alle restlichen, inneren Knoten aufgerufen. Für alle virtuellen Kanten, die nun betrachtet werden, und für die virtuellen Kanten im ersten Schritt, welche nicht in der Außenfläche liegen, ist die Wahl des linken Pols egal, da es für die Größe der Außenfläche unwichtig ist, in welche der beiden zur virtuellen Kante adjazenten Flächen expandiert wird.

Die Reihenfolge, in der die Kanten eines inneren Knotens betrachtet werden, kann nicht vorherbestimmt werden. In Abbildung 3.13 ist ein Beispiel gegeben, in dem dies veranschaulicht wird. Angenommen, v_1 wird zuletzt betrachtet und die Kanten

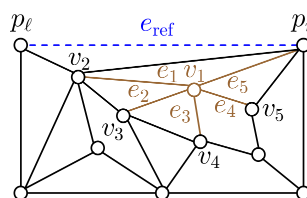


Abbildung 3.13: Einbettung der Kanten eines inneren Knotens in einer R-Komponente.

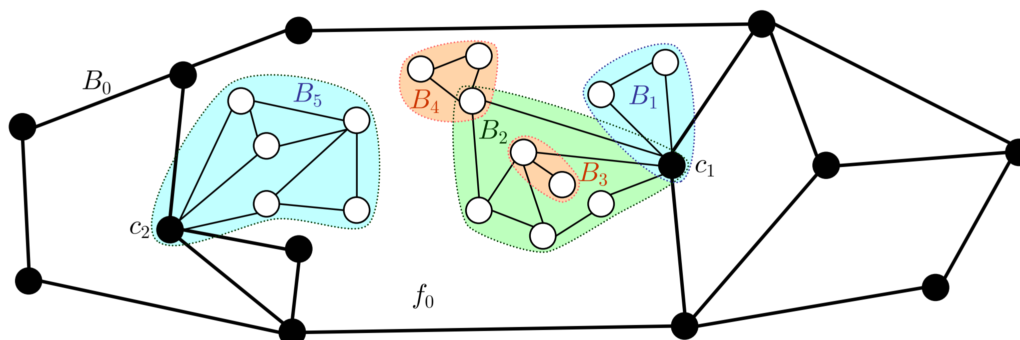


Abbildung 3.14: Eine fixe Einbettung Γ_0 von B_0 (dicke Linien), in die die Graphen G_{c_1,B_1}, G_{c_1,B_2} und G_{c_2,B_5} eingebettet wurden. (Quelle: [GM04b])

e_1 und e_3 sind virtuell. Wenn v_2 und v_4 betrachtet werden, wird die Einbettungsfunktion für die zu e_1 und e_3 zugehörigen Knoten aufgerufen und es werden Kanten in die Adjazenzliste von v_1 hinzugefügt. Die richtige Reihenfolge kann jedoch nicht garantiert werden, ohne die restlichen zu v_1 inzidenten Kanten zu betrachten. Denn zwischen den Kanten, die durch e_1 in die Adjazenzliste eingetragen werden, und den Kanten, die durch e_3 eingetragen werden, müssen noch die Kanten e_2 bzw. e_4 und e_5 eingefügt werden.

Falls also eine virtuelle Kante $e \notin f_{\text{ext}}^{\mu}$, betrachtet wird, werden die Kanten inzident zum Knoten $v \in e$, welcher in diesem Schritt nicht betrachtet wird, nicht direkt in die Adjazenzliste eingetragen. Sie werden zwischengespeichert und erst bei der Betrachtung von v in seine Adjazenzliste eingetragen.

3.2.2 Zusammenhängende Graphen

Gegeben ist ein zusammenhängender Graph $G = (V, E)$ und gesucht ist eine planare Einbettung von G, welche eine Außenfläche maximaler Größe besitzt. Sei $\mathcal{B}$ der BC-Baum von G, B ein Block in G und $c \in B$ ein Schnittknoten. Durch das Löschen der Kante (c, B) in $\mathcal{B}$ wird $\mathcal{B}$ in zwei zusammenhängende Komponenten geteilt. Sei $\mathcal{B}_B$ die Komponente, welche B enthält. Der durch die Knoten und Kanten der Skelettgraphen der Knoten in $\mathcal{B}_B$ definierte Graph sei $G_{c,B}$.

Die Idee des Algorithmus ist wie folgt (vergl. Abbildung 3.14). Gegeben sei eine Einbettung Γ_0 eines Blocks B_0. Um Γ_0 zu einer Einbettung von G zu expandieren,

Funktion `EinbettungVonRKomponente(`*Knoten* v`)`

if $v = p_\ell^\mu$ oder $v = p_r^\mu$ **then**
    $e :=$ nachfolgende Kante der Referenzkante in der Adjazenzliste von v
else
    $e :=$ irgendeine zu v inzidente Kante
if $v = p_\ell^\mu$ **then**
    $a := b(p_\ell^\mu)$
else if $v = p_r^\mu$ **then**
    $a := b(p_r^\mu)$
forall Adjazenzlisteneinträge e' von v beginnend mit $e, e' \neq e_{\text{ref}}$ **do**
    if e' ist virtuell und e' wurde schon betrachtet **then**
        Füge $\phi_v(e')$ zu $A(v)$ an Stelle a ein und aktualisiere a.
    else
        if e' ist virtuell **then**
            $\nu :=$ der zugehörige Knoten von e'
            if e' ist in f_{ext}^μ **then**
                $p :=$ Knoten $v' \in e'$ mit $\text{pred}_{v'}(e')$ ist in f_{ext}^μ
            else
                $p := v$
            $p_\ell^\nu :=$ der Knoten zugehörig zu p in skeleton(ν)
            if zugehöriger Knoten von p_r^ν in skeleton(μ) gleich p_r^μ **then**
                $b(p_r^\nu) := b(p_r^\mu)$
        end
        $a :=$ `FügeKanteHinzu`(v, e', a)
        if $v' := \text{twin}_v(e')$ nicht in f_{ext}^μ **then**
            $\phi_{v'}(e') := A(v')$
            $A(v') := \emptyset$
        end
    end
end
if $v = p_\ell^\mu$ **then**
    $b(p_\ell^\mu) := a$
else if $v = p_r^\mu$ **then**
    $b(p_r^\mu) := a$

muss eine Einbettung $\Gamma_{c,B}$ mit c in der Außenfläche für jeden Graph $G_{c,B}$ mit $c \in B$, $B \neq B_0$, gefunden werden. $\Gamma_{c,B}$ muss in eine zu c adjazente Fläche in Γ_0 eingebettet werden. Eine Fläche f_0 in Γ_0 kann so groß wie möglich gemacht werden, indem alle $\Gamma_{c,B}$ mit $c \in f_0$ in f_0 eingebettet werden.

Definition 3.2 ($\mathbf{smf}_{B}(c)$) $\mathrm{smf}_B(c)$ *bezeichnet die Summe der Größe der maximalen Flächen, welche c enthalten, über alle $G_{c,B'}$ mit $c \in B'$ und $B' \neq B$.*

Wenn c kein Schnittknoten ist, ist $\mathrm{smf}_B(c)$ gleich 0. Sei die Knotenlänge von c in B_0 gleich $\mathrm{smf}_B(c)$, falls c ein Schnittknoten ist, und seien die Kantenlängen für alle Kanten $e \in E_B$ gleich 1. Mit dem Algorithmus aus Kapitel 3.2.1 wird die Einbettung für Knoten B berechnet.

Theorem 3.1 *Für jeden Block B von G sei $\mathrm{smf}_B(v)$ die Länge der Knoten $v \in B$. Sei $B_{\max}$ der Block mit der Einbettung $\Gamma_{\max}$ mit der größten Fläche $f_{\max}$ über alle Blöcke. Dann kann $\Gamma_{\max}$ expandiert werden zu einer planaren Einbettung von G mit maximaler Außenfläche f. Die Fläche f entsteht durch das Einbetten jedes Graphen $G_{c,B'}$ mit $c \in B_{\max}$ und $B' \neq B_{\max}$ mit einer maximalen Außenfläche, welche c enthält, und anschließendem Hinzufügen der Einbettung zu $f_{\max}$.*

Algorithmus 3.9 zeigt, wie eine planare Einbettung mit maximaler Außenfläche berechnet werden kann. Die Variable $\mathrm{length}_B(c)$ speichert die Länge von Knoten c in Block B und $\mathrm{cstrLength}(B, c)$ wird auf die Größe der maximalen Fläche von $G_{c,B}$ gesetzt, welche c enthält. Die Funktion `BottomUpTraversal`(B, c) berechnet eine maximale Fläche in $G_{c,B}$, welche c enthält, und wird für jede Kante $(c, B) \in \mathcal{B}$ aufgerufen.

Algorithmus 3.9 : Planare Einbettung mit maximaler Außenfläche

$\mathcal{B}$:= BC-Baum von G, die Wurzel sei ein beliebiger Block B_r
forall $v \in B_r$ **do**

$$\mathrm{length}_{B_r}(v) := \sum_{(v,B)\in\mathcal{B}} \texttt{BottomUpTraversal}(B, v)$$

end
(B, ℓ) := `TopDownTraversal`(B_r)
return (B, ℓ)

Die Funktion `TopDownTraversal` durchläuft $\mathcal{B}$ rekursiv von der Wurzel zu den Blättern. Wenn `TopDownTraversal` für einen Block B aufgerufen wird, wurde die Länge für jeden Knoten $v \in B$ schon auf den Wert $\mathrm{smf}_B(v)$ gesetzt. Da die Funktion am Rekursionsende den Block B^* mit der maximalen Fläche der Größe ℓ^* zurückgibt, gilt $B^* = B_{\max}$ (vergl. Theorem 3.1). Um alle $G_{c,B}$ mit $c \in B^*$ und $B \neq B^*$ mit einer maximalen Außenfläche, welche c enthält, einzubetten, muss $\mathcal{B}$ rekursiv, startend in Block B^*, durchlaufen werden. Der Algorithmus berechnet die Größe einer maximalen Außenfläche von G in Zeit $O(|V| + |E|)$.

Funktion `BottomUpTraversal` (*Block B, Schnittknoten c*)

forall $v \in B, v \neq c$ **do**
  $\text{length}_B(v) := \sum_{(v,B') \in \mathcal{B}}$ `BottomUpTraversal`(B', v)
end
$\text{length}_B(c) := 0$
$\text{cstrLength}(B, c) :=$ Größe der maximalen Fläche in B, welche c enthält
return $\text{cstrLength}(B, c)$

Funktion `TopDownTraversal` (*Block B*)

$(B^*, \ell^*) := (B,$ Größe einer maximalen Fläche in $B)$
forall $(B, c) \in \mathcal{B}$ für die ein $(c, B') \in \mathcal{B}, B' \neq B$ existiert **do**
  $\text{cstrLength}(B, c) :=$ Größe einer maximalen Fläche in B, welche c enthält
  $L := \sum_{\{B',c\} \in \mathcal{B}} \text{cstrLength}(B', c)$
  forall $(c, B') \in \mathcal{B}$ **do**
    $\text{length}_{B'}(c) := L - \text{cstrLength}(B', c)$
    $\ell' :=$ `TopDownTraversal`(B')
    if $\ell' > \ell$ **then** $(B^*, \ell^*) := (B', \ell')$
  end
end
return (B^*, ℓ^*)

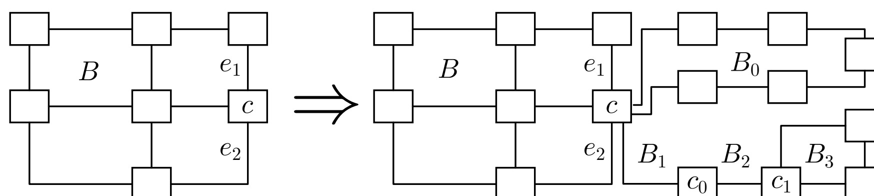


Abbildung 3.15: Zusammenfügen verschiedener Block-Einbettungen zu Γ_G.

Um eine planare Einbettung von G mit maximaler Außenfläche zu berechnen, wird ein Block gewählt, welcher eine Fläche maximaler Größe enthält. Für diesen Block wird mit dem Algorithmus aus Kapitel 3.2.1 eine planare Einbettung berechnet und zu Γ_G hinzugefügt. Die Außenfläche des Blocks wird als Außenfläche der planaren Einbettung von G übernommen. Anschließend wird für alle Schnittknoten c in B rekursiv für alle zu c adjazenten Blöcke $B' \neq B$ eine planare Einbettung berechnet. Falls c in der Außenfläche von Γ_B liegt, werden die Einbettungen $\Gamma_{B'}$ zur Außenfläche hinzugefügt, ansonsten werden sie in eine beliebige Fläche eingefügt. Das Einfügen geschieht, indem die Adjazenzlisten für alle Knoten $v \neq c$ in Γ_G kopiert werden. Für c müssen die Adjazenzlisteneinträge zwischen die beiden Kanten eingefügt werden, die in der Fläche liegen, in welche der Block eingebettet werden soll. Dies wird anhand eines Beispiels in Abbildung 3.15 veranschaulicht. Für die zu c adjazenten Blöcke B_0 und B_1 wurde rekursiv eine planare Einbettung berechnet. Die Einbettungen sollen nun mit Γ_B zu Γ_G vereinigt werden. c liegt in der Außenfläche und die Adjazenzlisteneinträge für c in den rekursiv erzeugten planaren Einbettungen für B_0 und B_1 werden zwichen die Kanten e_1 und e_2 eingefügt.

3.3 Minimale erweiterte Tiefe

Sei $G = (V, E)$ ein planarer Graph und B ein Block in G mit planarer Einbettung Γ_B. Eine Erweiterung (engl.: extension) von Γ_B bezeichnet eine planare Einbettung von G, welche durch das Einbetten von allen Graphen $G_{c,B'}$ mit $c \in B$ und $B' \neq B$ entsteht, so dass c in der Außenfläche liegt und diese Graphen in eine Fläche von Γ_B eingebettet werden.

Sei $m_{c,B'}$ die minimale Tiefe einer planaren Einbettung von $G_{c,B'}$ mit c in der Außenfläche. Sei außerdem:

$$m_B(c) := \max\big(\{0\} \cup \{m_{c,B'} \mid c \in B', B' \neq B\}\big) \tag{3.7}$$

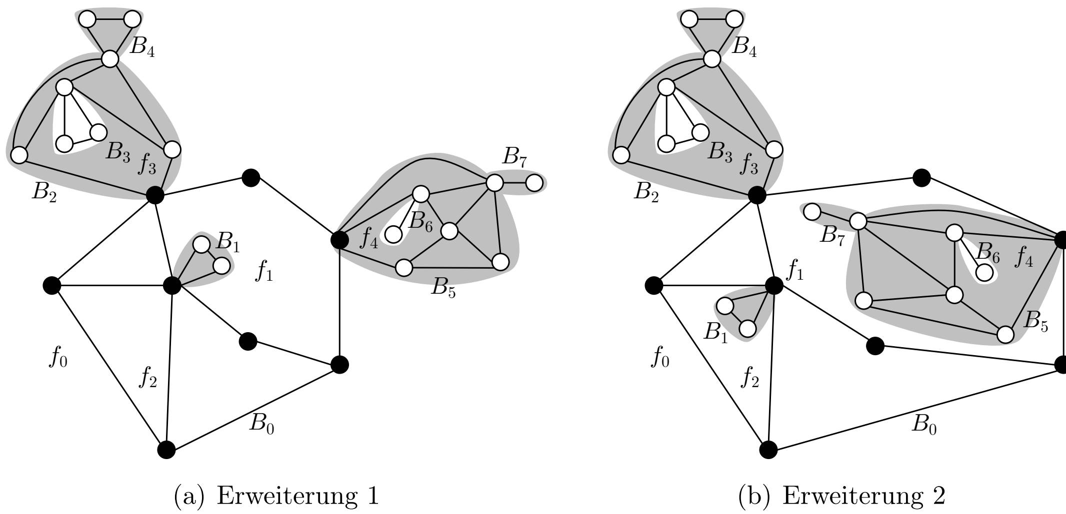
$$m_B := \max_{c \in B}\big(m_B(c)\big) \tag{3.8}$$

$$M_B := \{c \in B \mid m_B(c) = m_B\} \tag{3.9}$$

Die minimale Tiefe einer Erweiterung einer Einbettung von B ist m_B, falls eine Einbettung von B existiert, so dass alle Kanten in M_B in einer gemeinsamen Fläche liegen. Ansonsten beträgt die mnimale Tiefe der Erweiterung einer Einbettung $m_B +$ 2 (vergl. Abbildung 3.16). Im folgenden Lemma wird der Zusammenhang zwischen minimaler Tiefe und maximaler Außenfläche gezeigt.

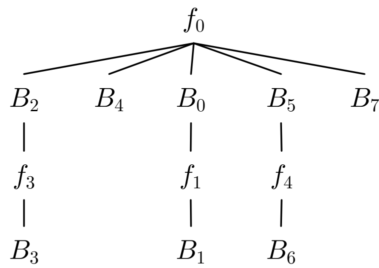
Lemma 3.2 *Sei die Länge aller Kanten in einem Block B gleich 0 und die Länge eines Knotens $v \in B$ gleich 1, wenn $v \in M_B$ und 0 sonst. Sei (Γ^*, f^*) die planare Einbettung von B mit maximaler Außenfläche f^*. Sei außerdem $\mathcal{E}_B$ die Menge aller Erweiterungen von B. Dann ist die minimale Tiefe einer Erweiterung einer Einbettung von B:*

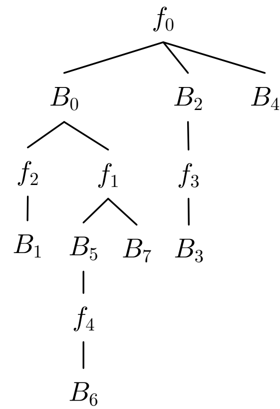
$$\min_{E \in \mathcal{E}_B}\big(\text{depth}(E)\big) := \begin{cases} m_B & \text{size}(f^*) = |M_B| \\ m_B + 2 & \textit{sonst} \end{cases} \tag{3.10}$$



(a) Erweiterung 1

(b) Erweiterung 2





(c) erweiterter dualer BC-Baum von Erweiterung 1 $\Rightarrow$ Tiefe 3

(d) erweiterter dualer BC-Baum von Erweiterung 2 $\Rightarrow$ Tiefe 5

Abbildung 3.16: Zwei verschiedene Erweiterungen einer Einbettung Γ_0 eines Blocks B_0. (Quelle: [GM04b])

Algorithmus 3.12 berechnet eine planare Einbettung eines Graphen $G_{c,B}$ mit minimaler Tiefe, so dass c in der Außenfläche liegt. Als erstes wird der BC-Baum $\mathcal{B}$ von den Blättern beginnend zur Wurzel durchlaufen und die Werte $m_{c,B}$ für alle Kanten (c, B) in $\mathcal{B}$ berechnet. Dabei werden die Längen der Knoten $v \neq c$ in B nach Lemma 3.2 gesetzt und eine maximale Fläche in B berechnet, welche c enthält.

Anschließend wird $\mathcal{B}$ von der Wurzel zu den Blättern durchlaufen. Wenn ein Block $B = (V_B, E_B)$ betrachtet wird, sind die Werte $m_{c,B'}$ für jeden Schnittknoten c in G mit $c \in B$ und jeden Block $B' \neq B$ schon berechnet. Nach Lemma 3.2 werden die Knotenlängen in B gesetzt und die minimale Tiefe einer Erweiterung von B durch das Finden einer maximalen Fläche in B berechnet. Bevor die Kinder von B im BC-Baum $\mathcal{B}$ betrachtet werden, werden die Werte $m_{c,B}$ für alle Kanten $(B, c) \in \mathcal{B}$ berechnet. Dabei werden zwei Fälle unterschieden:

1. $M_B = \{c_1, \ldots, c_k\}$ mit $k \geq 2$: Es wird eine maximale Fläche in Block B berechnet, welche c enthält. Dabei werden die schon zugewiesenen Knotenlängen benutzt.

2. $M_B = \{c\}$: In diesem Fall können die Knotenlängen nicht weiter benutzt werden, da $m_2 = \max_{v \in B, v \neq c} m_B(v)$ kleiner ist als m_B. Dieser Fall kann allerdings nur einmal für jeden Block auftreten. Es werden neue Knotenlängen gemäß $M_2 = \{c \in V_B \backslash \{v\} \mid m_B(c) = m_2\}$ und eine maximale Fläche in B, welche c enthält, mit diesen Knotenlängen berechnet. So wird sichergestellt, dass versucht wird, alle Blöcke, die Tiefe m_2 besitzen und im BC-Baum Kinder von c sind, in die Außenfläche einzubetten.

Algorithmus 3.12 : Planare Einbettung mit minimaler Tiefe

$\mathcal{B}$:= BC-Baum von G, die Wurzel sei ein beliebiger Block B_r
forall $v \in B_r$, $(v, B') \in \mathcal{B}$, $B' \neq B_r$ **do**
  $m_{B'}(v)$:= `BerechneMinTiefe`(v, B')
end
return `MinimaleTiefeRekursion`(B_r)

Der Algorithmus 3.12 berechnet in linearer Zeit die minimale Tiefe von G. Eine planare Einbettung mit minimaler Tiefe wird analog zu einer planaren Einbettung mit maximaler Außenfläche (vergl. Kapitel 3.2) berechnet, wobei die Kantenlängen alle 0 sind und die Knotenlängen wie in diesem Kapitel beschrieben gesetzt werden.

3.3.1 Modifikation des Algorithmus für minimale Tiefe

Auch wenn im Paper [GM04b] die Definition der minimalen Tiefe wie in Definition 2.34 gegeben ist, berechnet der beschriebene Algorithmus effektiv eine Einbettung mit minimaler erweiterter Tiefe nach Definition 2.37.

Der Algorithmus behandelt Blöcke mit nur einer Kante, welche also keine eigene Fläche besitzen, genauso wie alle anderen Blöcke. Da sie aber keine eigene Fläche

Funktion `BerechneMinTiefe(`*Schnittknoten c, Block B*`)`

$m_B := 0$
$M_B := \emptyset$
forall $v \in B$, $(v, B') \in \mathcal{B}$, $B' \neq B$ **do**
    $m_{B'}(v) :=$ `BerechneMinTiefe(`v, B'`)`
    if $m_B < \text{length}(v, B')$ **then**
        $m_B := \text{length}(v, B')$
        $M_B := \{v\}$
    else if $m_B = \text{length}(v, B')$ **then**
        $M_B := M_B \cup \{v\}$
end
forall $v \in B$ **do**
    if $v \in M_B$ **then**
        $\text{length}(v) := 1$
    else
        $\text{length}(v) := 0$
end
$\ell :=$ Größe einer maximalen Fläche in B, welche c enthält
if $\ell = |M_B|$ **then**
    return m_B
else
    return $m_B + 2$

besitzen, haben sie keinen Einfluss auf den dualen Graphen und damit auch nicht auf den erweiterten dualen BC-Baum — und auch nicht auf die Tiefe der Einbettung.

In Abbildung 3.17 ist ein Beispiel dargestellt, bei dem der Algorithmus die falsche Tiefe berechnen würde. Es soll bestimmt werden, welche minimale Tiefe für Block B_0 möglich ist, wenn e in der Außenfläche liegt. Der Algorithmus würde berechnen, dass alle Schnittknoten in B_0 jeweils nur zu einem Block mit Tiefe 0 adjazent sind. Damit ist $m_B = 0$ und die Menge $M_B = \{a, b, c, d\}$. Es würden dann alle Kantenlängen auf 0 und die Knotenlänge für a, b, c und d gleich 1 gesetzt werden. Mit diesen Werten würde eine maximale Außenfläche, welche e enthält, berechnet. Die Außenfläche hätte Größe 3, und wäre damit ungleich $|M_B|$. Der Algorithmus würde also bestimmen, dass B_0 eine minimale Tiefe von $m_B + 2 = 2$ hätte. Richtig wäre jedoch Tiefe 1.

Um diesen Fehler zu beheben, wird das Rekursionsende durch eine leere Menge M_B definiert. In diesem Fall wird die Anzahl Kanten im Block überprüft, ist sie genau 1, so wird die Tiefe 0 zurückgegeben, ansonsten Tiefe 1. Weiterhin wird das Hinzufügen zur Menge M_B für Schnittknoten, welche nur zu Blöcken mit Tiefe 0 adjazent sind, verboten. Durch diese Modifikationen berechnet der Algorithmus eine Einbettung mit minimaler Tiefe.

Funktion `MinimaleTiefeRekursion`(*Block B*)

```
m_B := 0; M_B := ∅
forall v ∈ B, (v, B') ∈ 𝓑 oder (B', v) ∈ 𝓑 do
    if m_B < m_B'(v) then  m_B := m_B'(v); M_B := {v}
    else if m_B = m_B'(v) then  M_B := M_B ∪ {v}
end
forall v ∈ B do
    if v ∈ M_B then  length(v) := 1
    else  length(v) := 0
end
forall (B, v) ∈ 𝓑 do
    if |M_B| = 1 und parent(B) ∈ M_B then
        m2 := 0; M2 := ∅
        forall v ∈ B, (v, B') ∈ 𝓑, v ∉ M_B do
            if m2 < m_B'(v) then  m2 := m_B'(v); M2 := {v}
            else if m2 = m_B'(v) then  M2 := M_B ∪ {v}
        end
        forall v ∈ B do
            if v ∈ M_B then  length'(v) := 1
            else  length'(v) := 0
        end
        ℓ := Größe einer maximalen Fläche in B, welche c enthält, mit Kno-
              tenlängen length'(v) für alle v in B
    else
        ℓ := Größe einer maximalen Fläche in B, welche c enthält, mit Kno-
              tenlängen length(v) für alle v in B
    if |M_B| = 0 then  m_B(v) = 1
    else
        if ℓ = |M_B| then  m_B(v) = m_B
        else  m_B(v) = m_B + 2
    end
end
forall v ∈ B, (v, B') ∈ 𝓑 do
    minDepth(B) := MinimaleTiefeRekursion(Block B')
end
ℓ := Größe einer maximalen Fläche in B
if ℓ = |M_B| then
    return m_B
else
    return m_B + 2
```

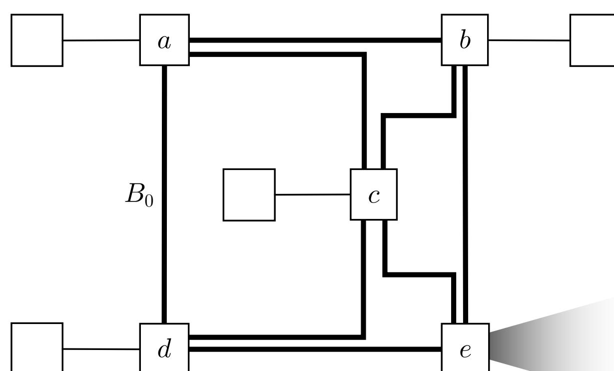


Abbildung 3.17: Einfluss von Blöcken mit nur einer Kante auf die Tiefe.

3.4 Minimale Tiefe und Maximale Außenfläche

Da die beiden Algorithmen für minimale Tiefe (Kapitel 3.3) und maximale Außenfläche (Kapitel 3.2) ähnlich sind, lassen sie sich zu einem neuen Algorithmus kombinieren, welcher in linearer Zeit eine planare Einbettung mit maximaler Außenfläche über alle planaren Einbettungen mit minimaler erweiterter Tiefe berechnet. Dazu wird als Längenattribut das Paar (d, ℓ) eingeführt. Der Parameter d ist eine 0/1-Variable, welche im Algorithmus für minimale erweiterte Tiefe benutzt wird und Parameter ℓ stellt das Längenattribut dar, das im Algorithmus für maximale Außenfläche benutzt wird. Für die Tupel ist eine lineare Ordnung definiert:

$$(d, \ell) > (d', \ell') \quad \iff \quad d > d' \text{ oder } (d = d' \text{ und } \ell > \ell') \tag{3.11}$$

Immer, wenn eine maximale Fläche in einem Block berechnet werden soll, wird als erstes eine maximale Fläche nach der oben definierten linearen Ordnung berechnet. Wenn die erhaltene Fläche Größe (d^*, ℓ^*) hat und d^* die Anzahl der Schnittknoten ist, welche in eine bekannte Fläche eingebettet werden sollen, das heißt $d^* = |M_B|$, so wurde eine planare Einbettung mit maximaler Fläche über alle planaren Einbettungen mit minimaler Tiefe gefunden. Anderenfalls ist d^* irrelevant, da alle Erweiterungen dieselbe Tiefe haben. Daher wird der Algorithmus zum Finden einer maximalen Fläche in einem Block erneut aufgerufen, dieses Mal nur unter Berücksichtigung der zweiten Komponente ℓ als Längenattribut.

4 Einbettung von inneren Blöcken

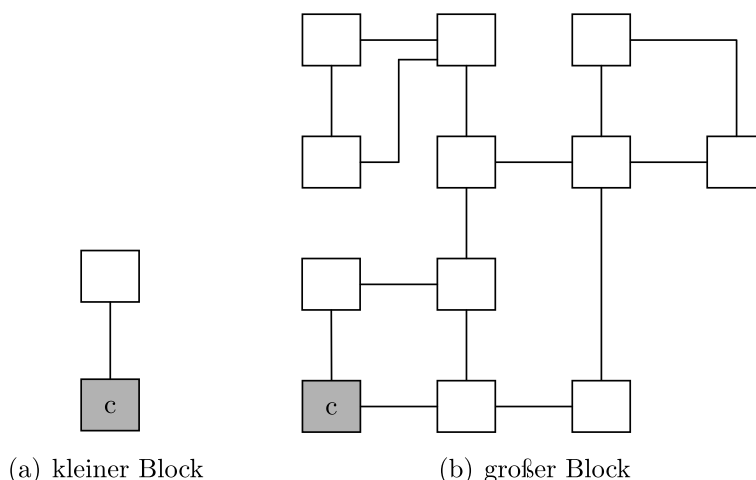
Die Algorithmen, die in Kapitel 3 vorgestellt wurden, beschreiben nur, wie die Außenfläche des Graphen und der Blöcke aussieht. Die Algorithmen beschreiben aber nicht, wie die restlichen Kanten eingebettet werden sollen. Genauso wird die Einbettung von Blöcken nicht beschrieben, welche nicht in die Außenfläche des Graphen bzw. des Blockes, zu dem sie adjazent sind und welcher vom Algorithmus zuerst betrachtet wird, eingebettet werden können.

In diesem Kapitel werden verschiedene Algorithmen vorgestellt, die angeben, in welche innere Fläche ein Block eingebettet werden soll. Dabei wird davon ausgegangen, dass die Einbettung der Zweizusammenhangskomponenten fix gegeben ist. Wie schon erwähnt, wird in den Algorithmen aus Kapitel 3 für einen inneren Block nicht definiert, in welche Fläche er eingebettet werden soll. In Kapitel 4.1 ist beschrieben, wie die Fläche für diese Blöcke in der ersten Implementierung der Algorithmen gewählt wurde. Die Kapitel 4.2 und 4.3 stellen dann verschiedene neue Lösungsansätze vor.

In Kapitel 4.4 wird ein Algorithmus beschrieben, der das dort neu eingeführte Einbettungsmaß der lexikografisch größten Schichtenmenge optimiert. Dazu bestimmt der Algorithmus sowohl die Einbettung von Kanten, die nicht in der Außenfläche liegen, als auch die Wahl der inneren Fläche, in die ein Block eingebettet werden soll, der nicht in der Außenfläche liegt. In Kapitel 5.3.1 werden alle Algorithmen dieses Kapitels experimentell miteinander verglichen.

4.1 Simpler Ansatz

Bei der ersten Implementierung der Algorithmen wurde der folgende Ansatz gewählt. Gegeben sei ein Block B und eine planare Einbettung für diesen Block. Für alle Schnittknoten c, welche nicht in der Außenfläche der Einbettung liegen, wird randomisiert eine adjazente Fläche gewählt und alle zu c adjazenten Blöcke $B' \neq B$ in diese Fläche eingebettet. Dieser Ansatz ist sehr simpel und auch dementsprechend effizient in der Laufzeit. Er soll als Vergleich für die anderen Algorithmen, welche in diesem Kapitel vorgestellt werden, dienen. Es soll dadurch ersichtlich werden, ob eine bestimmte Einbettungsstrategie Vorteile gegenüber einer Lösung bringt, welche keine Kriterien bei der Wahl der Einbettungsfläche beachtet.



(a) kleiner Block (b) großer Block

Abbildung 4.1: Vergleich der Größe zweier Blöcke.

4.2 Kleine Flächen

Das Ziel dieser Variante ist, dass die inneren Flächen möglichst klein sind. Das Problem lässt sich formal wie folgt definieren. Gegeben sei ein Block B', eine Menge von Blöcken $\mathcal{B} = \{B_0, \ldots, B_m\}$ und für jeden Block B_i ein Schnittknoten c_i, welcher B_i mit B' verbindet. Für B' ist eine planare Einbettung bekannt, die inneren Flächen seien $\mathcal{F} = \{f_0, \ldots, f_n\}$, wobei jeder Schnittknoten c_i in mindestens einer Fläche f_j liegt. Gesucht ist eine Zuordnung für jeden Block $B_j \in \mathcal{B}$ zu einer Fläche $f_i \in \mathcal{F}$, sodass die maximale Anzahl an Blöcken, die einer Fläche zugeordnet werden, minimal ist.

Da nicht jeder Block gleich groß ist und ein Block noch zu weiteren Blöcken adjazent sein kann, wird das Problem noch erweitert. Ohne diese Erweiterung würden beide Blöcke in Abbildung 4.1 gleich behandelt, obwohl sie einen unterschiedlich starken Einfluss auf die Größe der Fläche haben, in die dieser Block eingebettet wird. Anstatt die maximale Anzahl an Blöcken, welche zu einer Fläche zugeordnet werden, als Bewertung zu nehmen, wird die Knotenanzahl in diesen Blöcken samt aller zu ihnen adjazenten Blöcke, außer B', berechnet und als Bewertungskriterium genommen (vergl. Gleichung (4.1)). Zwei Blöcke werden als adjazent zueinander bezeichnet, falls sie einen gemeinsamen Schnittknoten besitzen.

$$\rho_j = |V(B_j)| + \sum_{\substack{B_i \text{ adjazent zu } B_j \\ B_i \neq B'}} \rho_i \tag{4.1}$$

Es lässt sich eine Heuristik verwenden, welche in Algorithmus 4.1 dargestellt ist.

Dabei sei:

$$\chi(c, B) = |V(B)| + \sum_{\substack{B_k \text{ adjazent zu } B \\ \xi(B,B_k) \neq c}} \chi(\xi(B, B_k), B_k) \quad (4.2)$$

$$\xi(B, B_k) = \text{Schnittknoten, welcher } B \text{ und } B_k \text{ verbindet} \quad (4.3)$$

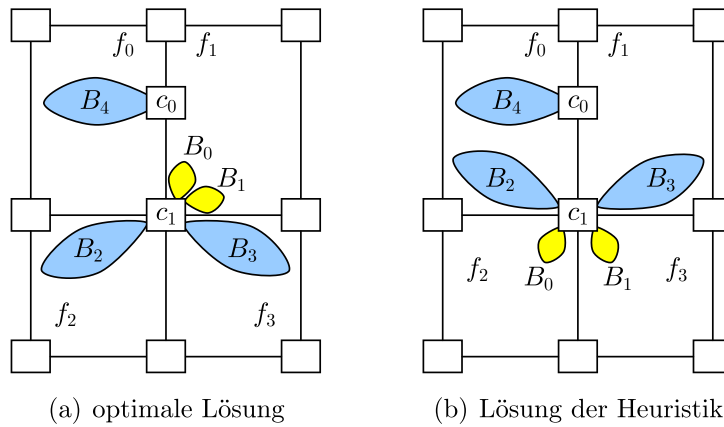
Algorithmus 4.1 : Heuristik für kleine innere Flächen

Eingabe: B, $\mathcal{B} := \{B_0, \ldots, B_m\}$, $\mathcal{F} := \{f_0, \ldots, f_n\}$
$\mathcal{B}' := \mathcal{B}$
$L :=$ Array, welches $\sum_{B \in \mathcal{B}} \chi(\xi(B', B), B)$ Mengen von Flächen speichert
$L[0] := \mathcal{F}$
for $j := 0,\ j < \sum_{B \in \mathcal{B}} \chi(\xi(B', B), B),\ j := j + 1$ **do**
  while $L[j] \neq \emptyset$ **do**
    $f :=$ ein Element aus $L[j]$
    $B :=$ ein Block aus $\mathcal{B}'$ adjazent zu f
    Bette Block B in Fläche f ein und entferne B aus $\mathcal{B}'$.
    if f adjazent zu einem $B \in \mathcal{B}'$ **then**
      $L[j + \chi(\xi(B', B), B)] := L[j + \chi(\xi(B', B), B)] \cup f$
    $L[j] := L[j] \setminus f$
  end
end

Es werden die Blöcke aus $\mathcal{B}$ nacheinander jeweils in eine Fläche, welche zu diesem Block adjazent ist und zu dem Zeitpunkt geringste Größe hat, eingebettet. Die Werte $\chi(c, B)$ lassen sich in linearer Zeit berechnen. Um insgesamt lineare Laufzeit zu erhalten wird ein Trick verwendet. Es wird ein Array L der Größe $\sum_{B \in \mathcal{B}} \chi(\xi(B', B), B)$ verwendet, wobei jeder Array-Eintrag $L[i]$ eine Liste mit allen Flächen speichert, in die bisher i Knoten eingebettet wurden. Zu Beginn sind alle Flächen, die zu mindestens einem Schnittknoten adjazent sind, in $L[0]$.

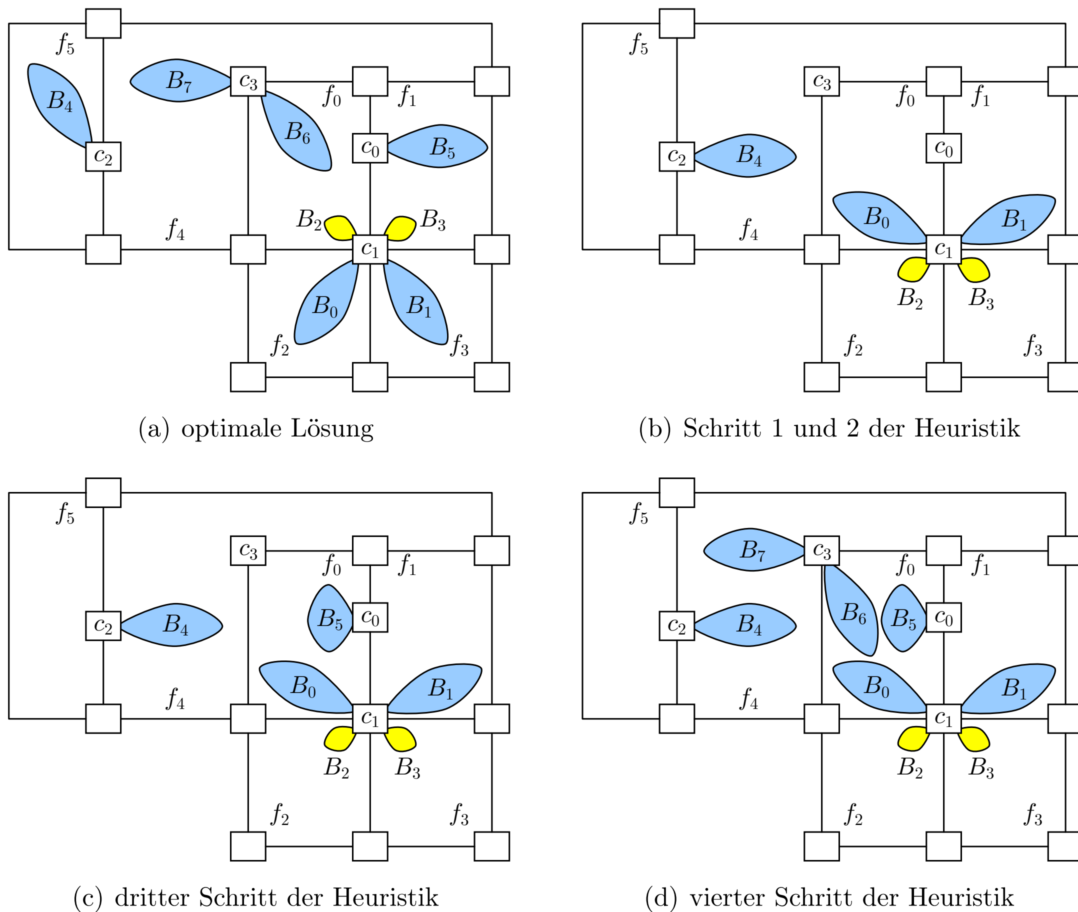
Es wird nun eine Schleife über die Variable j gestartet, beginnend mit j gleich 0. Falls $L[j]$ nicht leer ist, wird ein Block B, welcher noch nicht betrachtet wurde und der zur ersten Fläche f in $L[j]$ adjazent ist, in f eingebettet. Fläche f wird aus $L[j]$ entfernt und, falls f zu einem weiteren noch nicht betrachteten Block adjazent ist, in $L[j + \chi(\xi(B', B), B)]$ eingefügt. Falls $L[j]$ jedoch leer ist, wird j inkrementiert und die Schleife wiederholt, bis j die Array-Größe übersteigt. Dann ist das Ende der Schleife erreicht.

Die Anzahl der While-Schleifendurchläufe ist insgesamt durch $|\mathcal{B}|$ beschränkt, da in jedem Durchlauf ein Element aus $\mathcal{B}'$ entfernt wird und die While-Schleife nur Flächen, welche zu mindestens einem Block aus $\mathcal{B}'$ adjazent sind, betrachtet. Die Größe des Arrays L ist $O(|V|)$ und damit ist die Laufzeit insgesamt linear.



(a) optimale Lösung (b) Lösung der Heuristik

Abbildung 4.2: Güte der Heuristik für kleine innere Flächen.



(a) optimale Lösung (b) Schritt 1 und 2 der Heuristik

(c) dritter Schritt der Heuristik (d) vierter Schritt der Heuristik

Abbildung 4.3: Beispielgraph mit optimaler Lösungsgüte $n+1$ und Heuristiklösungsgüte $3n$. Die Blöcke B_1 und B_2 haben Größe 1, alle anderen Blöcke Größe n.

In Abbildung 4.2 ist ein Beispiel dargestellt, in dem die Heuristik nicht zwangsweise die optimale Lösung berechnet. Unter der Annahme, dass die Blöcke B_2, B_3 und B_4 Größe n und die Blöcke B_0 und B_1 Größe 1 haben, wäre die Lösung aus Abbildung 4.2(a) die optimale Verteilung der Blöcke auf die Flächen. Falls die Blöcke B_0 bis B_3 jedoch zuerst in der Heuristik betrachtet würden, könnte die Verteilung wie in Abbildung 4.2(b) aussehen. In der optimalen Lösung hätten die Flächen f_0, f_2 und f_3 Größe n und Fläche f_1 Größe 2. In der Lösung aus Abbildung 4.2(b) hätte Fläche f_0 hingegen Größe $2n$, die Flächen f_2 und f_3 Größe 1 und Fläche f_1 Größe n. Die optimalen Lösung hätte also eine maximale Flächengröße von n, die Lösung der Heuristik Größe $2n$. Durch geschicktes Einfügen weiterer Blöcke und Flächen lässt sich der Unterschied zwischen optimaler und Worst-Case Heuristik-Lösung sogar beliebig vergrößern, z.B. auf einen Unterschied von $n + 1$ zu $3n$ wie in Abbildung 4.3. In Kapitel 5.3.1 wird die Heuristik mit anderen Varianten verglichen, um zu sehen, wie sie sich trotz der schlechten theoretischen Worst-Case Güte in der Praxis verhält.

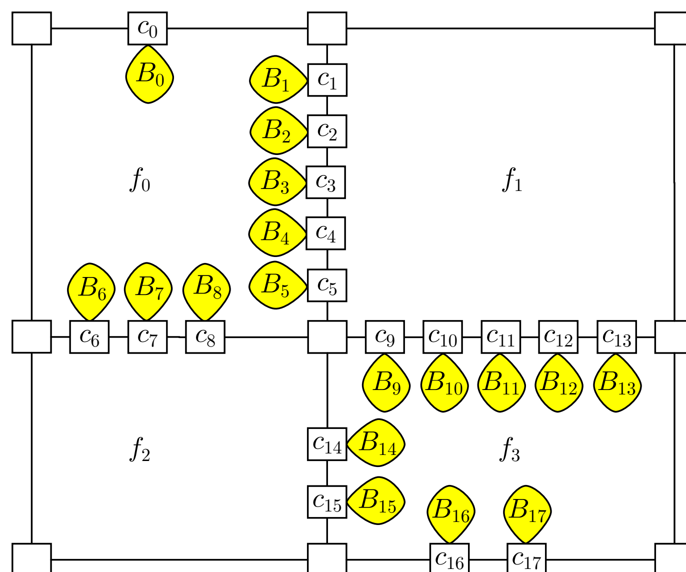
4.3 Wenige große Flächen

Eine andere Variante ist, die Blöcke, die in eine innere Fläche eingebettet werden müssen, in möglichst wenige Flächen einzubetten. Dadurch erhält man eine geringe Anzahl großer Flächen.

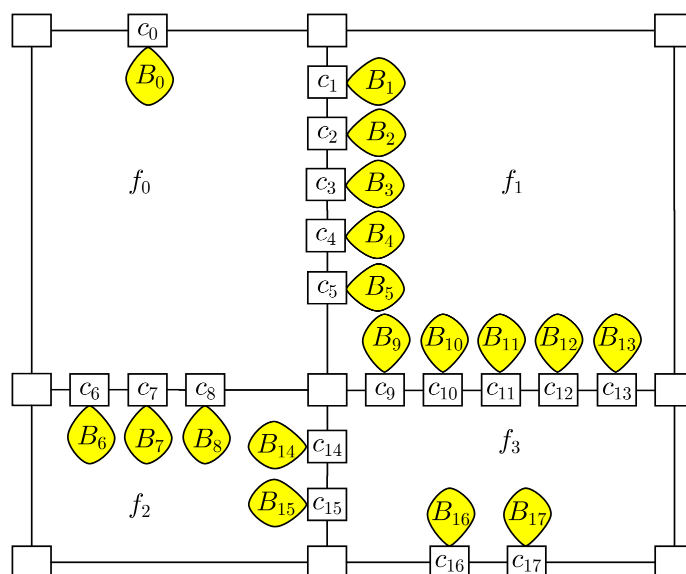
Formal definiert sich dies folgendermaßen: Gegeben sei ein Block $B' = (V', E')$, eine Menge von Blöcken $\mathcal{B} = \{B_0, \ldots, B_m\}$ und für jeden Block B_i ein Schnittknoten c_i, welcher B_i mit B' verbindet. Für B' ist eine planare Einbettung gegeben, die inneren Flächen seien $\mathcal{F} = \{f_0, \ldots, f_n\}$. Gesucht ist eine Zuordnung für jeden Block $B_j \in \mathcal{B}$ zu einer Fläche $f_i \in \mathcal{F}$, sodass die Anzahl der Flächen, denen mindestens ein Block zugeordnet wird, minimal ist.

In Algorithmus 4.2 ist eine Heuristik für das Problem dargestellt. Die Heuristik wählt jeweils die Fläche f aus, welche die maximale Anzahl noch nicht betrachteter Schnittknoten enthält, und bettet alle Blöcke aus $\mathcal{B}$ adjazent zu diesen Schnittknoten in f ein. Anschließend werden diese Blöcke aus $\mathcal{B}$ entfernt. Durch das geschickte Verwalten der Anzahl der noch nicht betrachteten Schnittknoten adjazent zu den Flächen kann, wie bei der Heuristik für kleine innere Flächen, garantiert werden, dass die While-Schleife insgesamt nur maximal $|\mathcal{B}|$-mal aufgerufen wird. Die Größe von L ist $O(|V|)$ und es ergibt sich auch hier eine lineare Laufzeit.

In dem Beispiel aus Abbildung 4.4 wäre die optimale Lösung, die Blöcke in die Flächen f_0 und f_3 einzubetten. Die Heuristik hingegen würde zunächst alle Blöcke adjazent zu den Schnittknoten in Fläche f_1 in diese einbetten, dann alle Blöcke in Fläche f_2, dann in f_3 und zuletzt in Fläche f_0. Die optimale Lösung würde also nur in zwei Flächen Blöcke einbetten, die Lösung der Heuristik jedoch in jede innere Fläche mindestens einen Block. In Kapitel 5.3.1 wird untersucht, wie gut die Ergebnisse der Heuristik in der Praxis sind.



(a) optimale Lösung



(b) Lösung der Heuristik

Abbildung 4.4: Güte der Heuristik für wenige große innere Flächen.

Algorithmus 4.2 : Heuristik zum Einbetten von inneren Blöcken in möglichst wenige Flächen

```
Eingabe: 𝓑 = {B_0, ..., B_m}, 𝓒 = {c_0, ..., c_m}, 𝓕 = {f_1, ..., f_n}
L := Array, welches |𝓒| Mengen von Flächen speichert
forall f ∈ 𝓕 do
    i := Anzahl Schnittknoten in f
    L[i] := L[i] ∪ f
end
𝓑' := 𝓑
𝓕' := 𝓕
for j := |𝓒|, j > 0, j := j − 1 do
    while L[j] ≠ ∅ do
        f := ein Element aus L[j]
        forall B_j ∈ 𝓑', c_j ∈ f do
            Bette Block B_j in Fläche f ein.
            Entferne B_j aus 𝓑'.
        end
        Entferne f aus 𝓕' sowie aus L[j].
        forall f' ∈ 𝓕' mit ∃c ∈ f' : c ∈ f do
            k := Index der Menge in L, welche f' enthält
            L[k] := L[k] \ f'
            L[k − 1] := L[k − 1] ∪ f'
        end
    end
end
```

4.4 Große äußere Schichten

In diesem Kapitel wird eine Heuristik vorgestellt, welche den Algorithmus für maximale Außenfläche aus Kapitel 3.2 erweitert. Es werden die dort berechneten Knoten- und Kantenlängen benutzt und die Berechnung der Einbettung um weitere Kriterien ergänzt. Diese Kriterien entscheiden, in welche Fläche eine virtuelle Kante expandiert werden soll (für zweizusammenhängende Graphen) oder in welche Fläche ein Block eingebettet wird (für zusammenhängende Graphen). Es werden zunächst die Begriffe einer Schicht und der Innenkreisgröße einer Schicht eingeführt.

Definition 4.1 (Schicht) *Eine Schicht $\mathcal{S}$ ist eine Teilmenge der Flächen, wobei Schicht $\mathcal{S}_i$ alle Flächen enthält, welche Abstand i zur Außenfläche haben.*

Definition 4.2 (Innenkreisgröße einer Schicht) *Die Innenkreisgröße einer Schicht $\mathcal{S}_i$ ist gleich der Anzahl der Kanten, welche sowohl in Schicht $\mathcal{S}_i$ als auch in Schicht $\mathcal{S}_{i+1}$ enthalten sind, plus der doppelten Anzahl der Brücken in den Flächen aus Schicht $\mathcal{S}_i$.*

Das Ziel ist es, die lexikografisch größte Schichtenaufteilung zu finden. Das bedeutet, es wird die Schichtenmenge $\{\mathcal{S}_0^*, \ldots, \mathcal{S}_m^*\}$ gesucht, so dass $\mathcal{S}_0^*$ die maximale Innenkreisgröße über alle möglichen Schichten $\mathcal{S}_0$ hat. Schicht $\mathcal{S}_0$ entspricht der Außenfläche des Graphen. Wird die Innenkreisgröße von $\mathcal{S}_0$ maximiert, so entspricht dies der Maximierung der Kanten der Außenfläche. Mit fest gewählter Schicht $\mathcal{S}_0^*$ wird eine Schicht $\mathcal{S}_1^*$ gesucht, welche über alle möglichen Schichten $\mathcal{S}_1$ die maximale Innenkreisgröße besitzt usw.

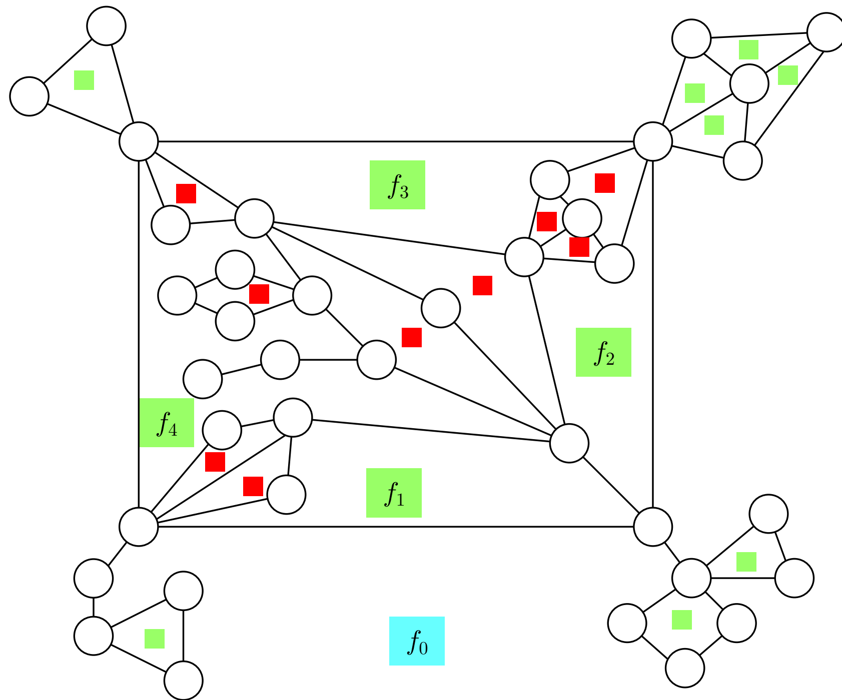
Abbildung 4.5 veranschaulicht dies. In beiden Einbettungen ist Schicht $\mathcal{S}_0 = \{f_0\}$ gleich und in blau dargestellt mit einer Innenkreisgröße von 27. Schicht $\mathcal{S}_1$ ist jeweils grün dargestellt und hat in Einbettung 1 eine Innenkreisgröße von 24 und in Einbettung 2 eine Innenkreisgröße von 14. Schicht $\mathcal{S}_2$, in rot, hat in Einbettung 1 eine Innenkreisgröße von 0 und in Einbettung 2 eine Innenkreisgröße von 8. In Einbettung 2 gibt es auch eine Schicht $\mathcal{S}_3$, in braun dargestellt, mit einer Innenkreisgröße von 0. Einbettung 1 ist lexikografisch größer als Einbettung 2, da bei ihr die Schicht $\mathcal{S}_1$ größer ist.

In Kapitel 4.4.1 wird beschrieben, wie das Problem für einen zweizusammenhängenden Graphen gelöst werden kann. Diese Lösung wird in Kapitel 4.4.2 benutzt, um eine Lösung für den allgemeinen Fall zu geben. Die Laufzeit des Algorithmus wird bewiesen und ein Beispiel dargestellt, in dem die Heuristik nicht zwangsweise die optimale Lösung berechnen würde. Das Problem ließe sich auch effizient exakt lösen, die vorgestellte Heuristiklösung hat jedoch den Vorteil, dass nur der Schritt, in dem die Einbettung berechnet wird, verändert wird. Dadurch lässt sich diese Variante leicht mit anderen Algorithmen kombinieren, ohne, dass dort die Berechnung der Knoten- und Kantenlängen verändert werden muss.

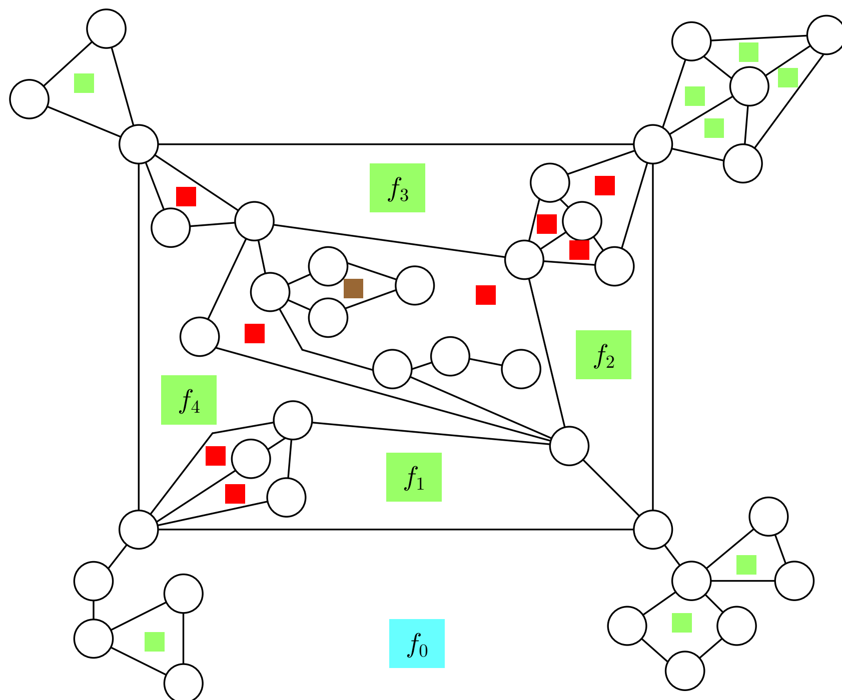
4.4.1 Zweizusammenhangskomponenten

Gegeben sei ein zweizusammenhängender Graph $G = (V, E)$, Knotenlängen $\ell(v_i)$ für $i = 1, \ldots, |V|$ und Kantenlängen $\ell(e_i)$ für $i = 1, \ldots, |E|$. Gesucht ist eine Einbettung Γ_G mit Außenfläche f_{ext} und lexikografisch größter Schichtenmenge. Der Algorithmus, der dieses Problem löst, benutzt die Datenstruktur der SPQR-Bäume (vergl. Kapitel 3.2.1). Die Wurzel des SPQR-Baums μ_r sei ein beliebiger Knoten, dessen Skelett eine Fläche f maximaler Größe mit einer realen Kante e_r enthält. Diese Fläche f wird als Fläche f_{ext} gewählt. Die Idee ist es, virtuelle Kanten zu expandieren. Für die maximale Außenfläche genügt es, alle virtuellen Kanten e, welche in der Außenfläche liegen, durch einen Weg p_e im Expansionsgraphen expansion(e) maximaler Länge zu ersetzen. Die restlichen virtuellen Kanten werden nach keinem festen System expandiert, das heißt es wird irgendeine zu dieser Kante adjazente Fläche als Expansionsfläche gewählt. In diesem Algorithmus wird die Wahl der Expansionsfläche und die Wahl der Einbettung einer P-Komponente gefällt.

Der Algorithmus fügt die Einbettung von skeleton(μ_r) zu Γ_G hinzu, wobei alle virtuellen Kanten jeweils in die Fläche expandiert werden, welche den geringsten Abstand zur Außenfläche f_{ext} besitzt. Für eine S-Komponente existiert nur eine Einbettung, welche übernommen wird, und für eine R-Komponente existieren genau

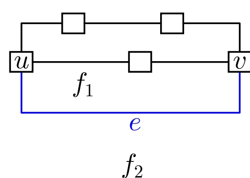


(a) Einbettung 1

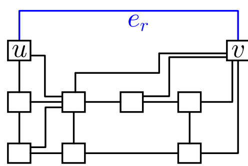


(b) Einbettung 2

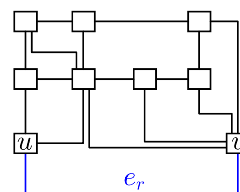
Abbildung 4.5: Zwei Einbettungen mit unterschiedlichen Schichten.



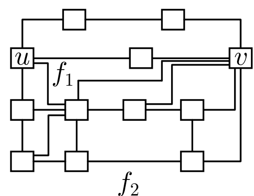
(a) Graph, in dem e expandiert werden soll.



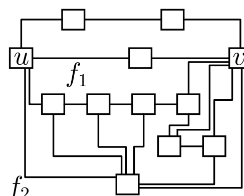
(b) Einbettung 1 für skeleton(μ).



(c) Einbettung 2 für skeleton(μ).



(d) Expandieren von e in Fläche f_2, Wahl von Einbettung 1.



(e) Expandieren von e in Fläche f_1, Wahl von Einbettung 2.

Abbildung 4.6: Wahl der Einbettung einer R-Komponente. e sei eine virtuelle Kante mit zugehörigem Knoten μ.

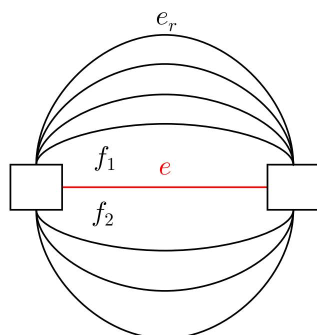
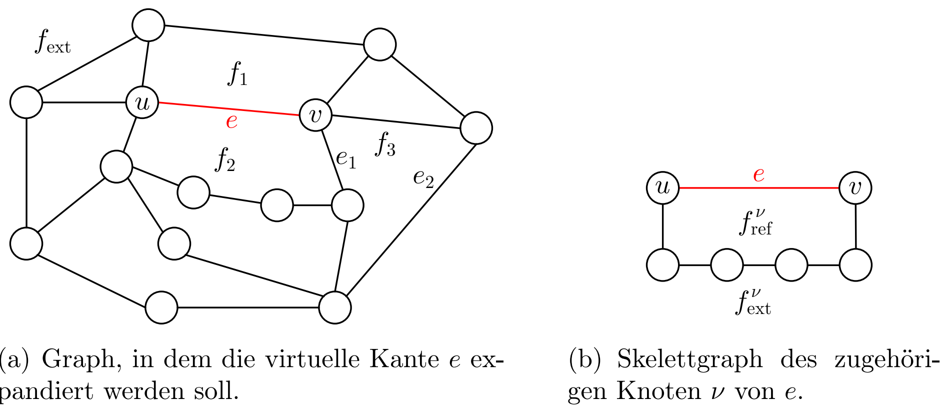


Abbildung 4.7: Einbettung einer P-Komponente.

zwei spiegelsymmetrische Einbettungen. Für μ_r ist es egal, welche der beiden Einbettungen einer R-Komponente genommen wird, für alle anderen SPQR-Baumknoten ist vorher bestimmt worden, in welche Fläche die Einbettung expandiert werden soll, so dass dadurch eindeutig gegeben ist, welche der beiden Einbettungen gewählt werden muss (vergl. Abbildung 4.6). Es wird immer die Fläche adjazent zu e_r bzw. der Referenzkante als Außenfläche des Skeletts gewählt, welche maximale Größe hat.

Falls μ_r ein P-Knoten ist, ist die Einbettung nicht eindeutig beschrieben. Die Wahl der Einbettung wird später detailliert beschrieben. Die Idee dabei ist es, die Kanten absteigend in der Reihenfolge ihrer Länge zu betrachten. Die Kanten werden nacheinander zwischen die schon betrachteten Kanten eingefügt und die Fläche als Expansionsfläche der Kante gewählt, welche den geringsten Abstand zur Außenfläche hat. Falls f_1 gewählt wurde, wird die nächste Kante unter dieser Kante eingebettet,



(a) Graph, in dem die virtuelle Kante e expandiert werden soll.

(b) Skelettgraph des zugehörigen Knoten ν von e.

Abbildung 4.8: Wahl der Expansionsfläche für eine virtuelle Kante.

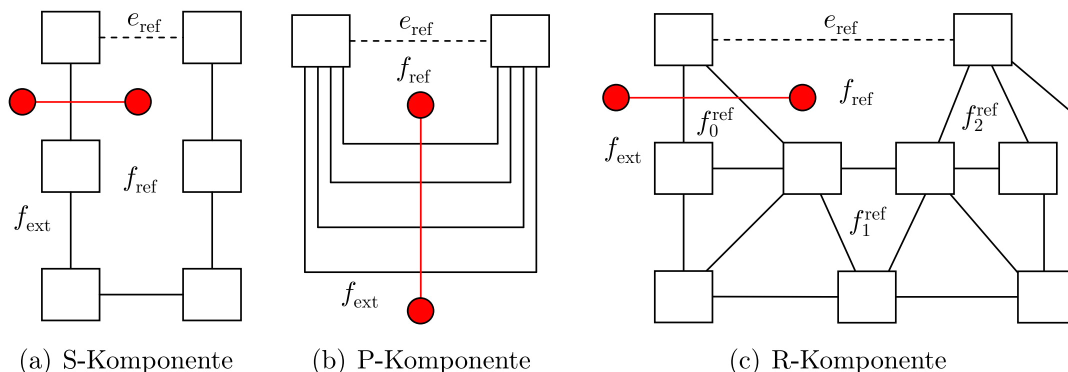
ansonsten darüber (vergl. Abbildung 4.7).

In dem Beispiel aus Abbildung 4.8 soll die virtuelle Kante e expandiert werden. Die beiden zu e adjazenten Flächen sind f_1 und f_2. Unter der Annahme, dass alle Kanten im Graphen reale Kanten mit Länge 1 sind, hat f_1 den Abstand 1 und f_2 den Abstand 2 zur Außenfläche. Dementsprechend würde e in f_1 expandiert werden und im Skelettgraphen von ν würde die Fläche f_{ref}^{ν} der Fläche f_2 und f_{ext}^{ν} der Fläche f_1 entsprechen. In den Werten $\delta_{\text{ref}}(\nu)$ und $\delta_{\text{ext}}(\nu)$ werden für die beiden zur Referenzkante adjazenten Flächen der Abstand zur Außenfläche f_{ext} gespeichert, wobei f_{ext}^{ν} die Fläche ist, in die e expandiert wird. In diesem Fall wäre $\delta_{\text{ref}}(\nu) = 2$ und $\delta_{\text{ext}}(\nu) = 1$. Seien die Kanten e_1 und e_2 virtuelle Kanten. Dann hätte der Weg von f_2 zu f_{ext} über die Fläche f_3 nicht zwangsweise Länge 2, da nicht garantiert werden kann, dass das Schneiden des Expansionsgraphen für e_1 bzw. e_2 nur eine Kantenkreuzung kostet (vergl. Abbildung 4.9). Es wird daher der Begriff der Dicke eines Skeletts im folgenden Kapitel definiert. Die Dicke eines Skeletts entspricht der minimalen Kantenkreuzungsanzahl für einen Skelettgraphen. Die Dicke des Skeletts des zugehörigen Knotens von e_1 würde dann der Länge von e_1 entsprechen, welche bei der Berechnung der Distanz von f_2 zu f_{ext} benutzt würde.

Dicke eines Skeletts

Die Dicke wird bei der Berechnung der Einbettung des Graphen benötigt. Da diese in einem Bottom-Up-Traversal des SPQR-Baums berechnet wird, ist bekannt, dass die Dicke eines Skeletts nur für die Kinder eines SPQR-Baumknotens benötigt wird. Das bedeutet, die Dicke von skeleton(μ_r) ist nicht von Interesse und in einem Skelett ist die Länge der Referenzkante, also die Dicke des Skeletts des zugehörigen Knotens der Referenzkante, auch nicht von Belang. Die Definition der Dicke entspricht dem Begriff „traversing cost", welcher in [GMW05] definiert und benutzt wird.

Sei $\ell^d(e)$ für die Kanten eines Skeletts skeleton(μ) eines SPQR-Baums $\mathcal{T}$ definiert als $\ell(e)$ für reale Kanten, 0 für die Referenzkante von μ und für eine virtuelle Kante e mit zugehörigem Knoten ν gleich $d(\nu)$. Der Begriff der Dicke $d(\mu)$ der Einbettung



(a) S-Komponente (b) P-Komponente (c) R-Komponente

Abbildung 4.9: Dicke eines Skelettgraphen.

eines Skeletts skeleton(μ) mit Referenzkante e_{ref} ist wie folgt definiert: Die Dicke gibt an, wie viele Kanten in G eine Linie schneiden muss, welche in der Fläche f_{ref} adjazent zur Referenzkante mit $f_{\text{ref}} \neq f_{\text{ext}}$ beginnt, um in die Außenfläche f_{ext} zu gelangen, ohne dabei die Referenzkante zu schneiden (vergl. Abbildung 4.9).

Die Dicke $d(\mu)$ ist anhand der Art von μ definiert als:

- S-Komponente: $d(\mu) = \min_{e \neq e_{\text{ref}}} \ell^d(e)$ (vergl. Abbildung 4.9(a))
- P-Komponente: $d(\mu) = \sum_{e \neq e_{\text{ref}}} \ell^d(e)$ (vergl. Abbildung 4.9(b))
- R-Komponente (vergl. Abbildung 4.9(c)): Seien $f_0^{\text{ref}}, \dots, f_k^{\text{ref}}$ die Flächen, welche mindestens eine Kante mit f_{ref} gemeinsam haben, außer f_{ext}. Es wird der duale Graph der Einbettung von skeleton(μ) berechnet, wobei die Kantenlänge zwischen zwei Flächen f_i, f_j, $i \neq j$, welche mindestens eine gemeinsame Kante besitzen, gleich dem minimalen $\ell^d(e)$ über alle gemeinsamen Kanten e ist. Für die Flächen $f_0^{\text{ref}}, \dots, f_k^{\text{ref}}$ sei $\ell(f_i^{\text{ref}})$ der minimale $\ell^d(e)$ Wert über alle gemeinsamen Kanten e von f_i^{ref} und f_{ref}. Weiterhin wird der Knoten zugehörig zu f_{ref} entfernt. Sei $\pi(f, f')$ ein kürzester Weg von f zu f' im dualen Graphen und $\ell(\pi(f, f'))$ seine Länge. Der Wert $d(\mu)$ entspricht dann:

$$d(\mu) = \min_{f_i^{\text{ref}} \in \{f_0^{\text{ref}}, \dots, f_k^{\text{ref}}\}} \left(\ell(\pi(f_{\text{ext}}, f_i^{\text{ref}})) + \ell(f_i^{\text{ref}}) \right)$$

Die Dicke aller Skelettgraphen lässt sich durch ein Bottom-Up-Traversal berechnen. Die Dicke des Skeletts des Wurzelknotens ist 0. Die Blätter enthalten außer der Referenzkante nur reale Kanten, somit sind alle $\ell^d(e)$-Werte gegeben. Im allgemeinen Fall wurde für alle Kinder die Dicke des Skelettgraphen schon berechnet, so dass auch dort alle $\ell^d(e)$-Werte gegeben sind. Für eine R-Komponente ist neben den $\ell^d(e)$-Werten noch wichtig, welche Fläche als f_{ext} gewählt wird. Dies sei die Fläche adjazent zu e_{ref} maximaler Größe. Die Größe ist die Summe der Kanten- plus

die Summe der Knotenlängen. Alle realen Kanten e haben Länge $\ell(e)$, für virtuelle Kanten sei die Kantenlänge gleich der Komponentenlänge (siehe Definition 3.1).

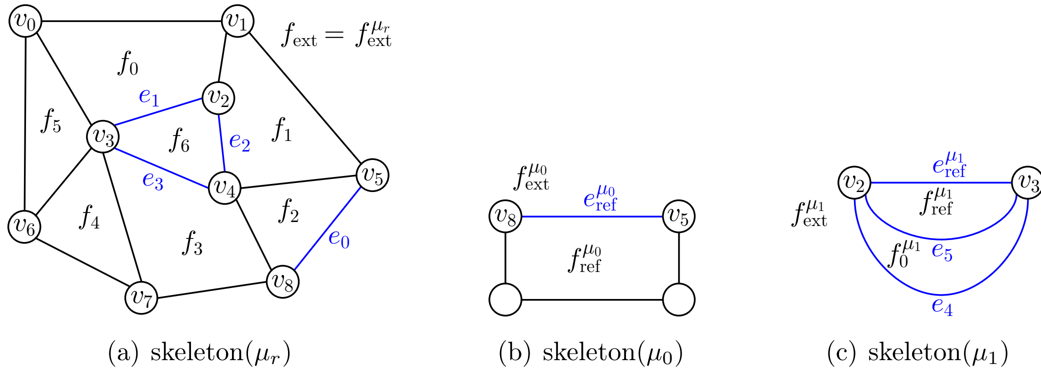
Die Laufzeit zur Berechnung der Dicken ist linear. Die Dicke wird für jeden Skelettgraph berechnet und, da die Größe von $\mathcal{T}$ inklusive seiner Skelettgraphen linear in der Größe von G ist, genügt es zu zeigen, dass jede Komponente nur lineare Zeit benötigt. Für eine S- und P-Komponente ist dies leicht nachvollziehbar. In einer R-Komponente kann das Berechnen des dualen Graphen in linearer Zeit geschehen, da es nur linear viele Flächen in einer Einbettung geben kann (linear in der Anzahl der Kanten) und jede Kante zu genau zwei Flächen adjazent ist. Das Bestimmen der Fläche f_{ext} geschieht durch Aufsummieren aller Kantenlängen in den zwei zur Referenzkante adjazenten Flächen und somit auch in linearer Zeit. Die Bestimmung der Kantenlänge im dualen Graphen kann auch in linearer Zeit geschehen. Es muss die maximale Kantenlänge aller zu den beiden adjazenten Flächen bestimmt werden. Jede Kante ist zu genau zwei Flächen adjazent und wird in der Berechnung daher auch nur zweimal betrachtet. Das Berechnen eines kürzesten Weges von f_{ext} zu den Knoten der Flächen $f'_0, \ldots, f'_k$ entspricht einem Single-Source-Shortest-Path (SSSP) Problem in einem ungerichteten Graphen mit positiven, ganzzahligen Kantenkosten. Dies kann in Zeit $O(m)$ gelöst werden [Tho99]. Dabei ist m die Anzahl der Kanten im dualen Graphen, diese ist linear in der Anzahl der Kanten im Skelett und somit ist die Gesamtlaufzeit für eine R-Komponente linear.

Berechnung der Einbettung

Sei $\mathcal{T}$ der SPQR-Baum von G mit Wurzel μ_r, wobei skeleton(μ_r) eine maximale Fläche mit einer realen Kante enthält. Die Dicke der Skelette von $\mathcal{T}$ wird wie im vorigen Kapitel beschrieben berechnet. Es werden nun in einem Top-Down-Traversal alle Knoten von $\mathcal{T}$ betrachtet und jeweils ihr Skelettgraph zu Γ_G hinzugefügt. Falls $\mu \neq \mu_r$ wird das Skelett von μ dazu in die Fläche $f_{\nu,\mu}$ expandiert. Diese wurde schon in dem Schritt berechnet, in dem der Elter ν von μ in $\mathcal{T}$ betrachtet wurde. Der Wert $\delta_{\text{ext}}(\nu)$ speichert den Abstand zwischen der Fläche, in die die virtuelle Kante e zugehörig zu ν expandiert werden soll, und f_{ext}. Der Wert $\delta_{\text{ref}}(\nu)$ hingegen entspricht dem Abstand der zweiten zu e adjazenten Fläche zu f_{ext}. Vor dem rekursiven Aufruf des Algorithmus werden $\delta_{\text{ref}}(\nu)$ und $\delta_{\text{ext}}(\nu)$ für alle Kinder von μ in $\mathcal{T}$ berechnet. Dadurch kann der minimale Abstand von einer Fläche f in einem Skelettgraphen skeleton(μ) zu f_{ext} berechnet werden, indem der Abstand $\pi(f, f^{\mu}_{\text{ref}})$ zu f^{μ}_{ref} und der Abstand $\pi(f, f^{\mu}_{\text{ext}})$ zu f^{μ}_{ext} in skeleton(μ) bestimmt wird. Der Weg darf dabei allerdings nicht die virtuelle Kante schneiden, die expandiert werden soll. Zu $\pi(f, f^{\mu}_{\text{ref}})$ wird dann noch der Abstand von f^{μ}_{ref} zu f_{ext} addiert und zu $\pi(f, f^{\mu}_{\text{ext}})$ der Abstand von f^{μ}_{ext} zu f_{ext}, das Minimum der beiden Summen entspricht dem minimalen Abstand von f zu f^{μ}_{ext}:

$$\pi(f, f_{\text{ext}}) = \min\{\pi(f, f^{\mu}_{\text{ref}}) + \delta_{\text{ref}}(\mu);\ \pi(f, f^{\mu}_{\text{ext}}) + \delta_{\text{ext}}(\mu)\} \quad (4.4)$$

In Abbildung 4.10 ist ein Beispiel dargestellt, in dem der Algorithmus eine Einbettung für einen Graphen berechnet, dessen SPQR-Baum einen R-Knoten μ_r als



(a) skeleton(μ_r) (b) skeleton(μ_0) (c) skeleton(μ_1)

Abbildung 4.10: Beispiel für Algorithmus große äußere Schichten. Der zugehörige Knoten für eine Kante e_i sei μ_i. Die Dicke der Skelette sei $d(\mu_0) = 1$, $d(\mu_1) = 4$, $d(\mu_2) = 3$, $d(\mu_3) = 2$, $d(\mu_4) = 3$ und $d(\mu_5) = 1$.

Wurzel hat. Zunächst wird das Skelett von μ_r zu Γ_G hinzugefügt und dann alle virtuellen Kanten in skeleton(μ_r) rekursiv expandiert:

- e_0 wird in f_{ext} expandiert (siehe Abbildung 4.10(b))
- e_1 wird in f_0 expandiert (siehe Abbildung 4.10(c))
- e_2 wird in f_1 expandiert mit $\delta_{\text{ref}}(\mu_2) = 3$ und $\delta_{\text{ext}}(\mu_2) = 1$
- e_3 wird in f_3 expandiert mit $\delta_{\text{ref}}(\mu_3) = 4$ und $\delta_{\text{ext}}(\mu_3) = 1$

In skeleton(μ_0) ist $\delta_{\text{ref}}(\mu_0) = 2$ und $\delta_{\text{ext}}(\mu_0) = 0$, in skeleton($\mu_1$) ist $\delta_{\text{ref}}(\mu_1) = 3$ und $\delta_{\text{ext}}(\mu_1) = 1$. Die virtuelle Kante e_4 wird in Fläche $f_{\text{ext}}^{\mu_1}$ expandiert mit $\delta_{\text{ref}}(\mu_4) = 4$ und $\delta_{\text{ext}}(\mu_4) = 1$. Wenn für die virtuelle Kante e_5 in skeleton(μ_1) entschieden werden soll, ob sie in $f_{\text{ref}}^{\mu_1}$ oder in $f_0^{\mu_1}$ expandiert wird, werden die jeweiligen Abstände der Flächen zu f_{ext} verglichen, und Fläche $f_{\text{ref}}^{\mu_1}$ gewählt:

$$\underbrace{\delta_{\text{ref}}(\mu_1)}_{=3} + \underbrace{\pi(f_{\text{ref}}^{\mu_1}, f_{\text{ref}}^{\mu_1})}_{=0} = 3 < 4 = \underbrace{\delta_{\text{ext}}(\mu_1)}_{=1} + \underbrace{\pi(f_0^{\mu_1}, f_{\text{ext}}^{\mu_1})}_{=\ell^d(e_4)=3} \qquad (4.5)$$

Die Berechnung der Einbettung wird nun formal für die verschiedenen Komponentenarten beschrieben. Zur Erinnerung: Vor dem rekursiven Aufruf für einen Knoten μ wurde die Fläche $f_{\nu,\mu}$ bestimmt, in die diese expandiert werden soll. Für die externe Fläche des Skeletts und die zweite zur Referenzkante inzidente Fläche wurde der Abstand zu f_{ext} in den Werten $\delta_{\text{ext}}(\mu)$ und $\delta_{\text{ref}}(\mu)$ abgespeichert. Die Berechnung der Einbettung unterscheidet anhand der Art des Skeletts.

S-Komponente. Es existiert nur eine mögliche Einbettung für μ und skeleton(μ) wird in Fläche $f_{\nu,\mu}$ expandiert. Diese Fläche wird auch als $f_{\mu,\mu'}$ für alle Kinder μ' von μ in $\mathcal{T}$ gewählt. Die Werte $\delta_{\text{ref}}(\mu)$ und $\delta_{\text{ext}}(\mu)$ können für alle Kinder μ' übernommen werden, d.h. $\delta_{\text{ref}}(\mu') = \delta_{\text{ref}}(\mu)$ und $\delta_{\text{ext}}(\mu') = \delta_{\text{ext}}(\mu)$.

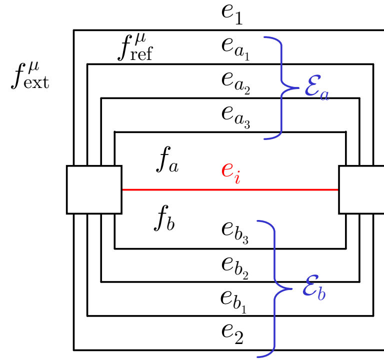


Abbildung 4.11: Einbettung der Kante e_i in einer P-Komponente.

P-Komponente. Der Skelettgraph besitzt m parallele Kanten $e_1, \ldots, e_m$. Falls μ der Wurzelknoten von $\mathcal{T}$ ist, existiert eine reale Kante e_1, welche in der Außenfläche liegt. Falls μ nicht die Wurzel ist, sei e_1 die Referenzkante von skeleton(μ). Die restlichen Kanten seien absteigend nach ihrem $\ell(e)$ Wert sortiert. $\ell(e)$ ist für reale Kanten die gegebene Kantenlänge von e und für virtuelle Kanten gleich der Komponentenlänge der Kante.

Die Kanten $e_2, \ldots, e_m$ werden nacheinander eingebettet. Die Kante e_1 sei schon in der Einbettung vorhanden, für sie existiert nur eine Einbettungsmöglichkeit, ebenso für e_2. Als Fläche $f_{\mu,\mu'}$, falls e_2 virtuell ist und μ' der zugehörige Knoten ist, wird die Außenfläche gewählt und e_2 zur Menge $\mathcal{E}_b$ hinzugefügt. Die nächste Kante, e_3, wird zwischen e_1 und e_2 eingebettet.

Es wird nun der Schritt betrachtet, in dem Kante e_i eingebettet werden soll (vergl. Abbildung 4.11). Die Position wurde im Schritt zuvor bestimmt. Falls e_i eine reale Kante ist, wird für die nächste Kante eine beliebige Position direkt über oder unter e_i gewählt. Ist e_i eine virtuelle Kante mit zugehörigem Knoten μ', muss zunächst entschieden werden, welche der beiden zu e_i adjazenten Flächen f_a oder f_b (siehe Abbildung 4.11) als $f_{\mu,\mu'}$ gewählt wird: Fläche f_a wird als Fläche $f_{\mu,\mu'}$ gewählt, falls

$$\delta_{\text{ref}}(\mu) + \sum_{e' \in \mathcal{E}_a} \ell^d(e') \quad \leq \quad \delta_{\text{ext}}(\mu) + \sum_{e' \in \mathcal{E}_b} \ell^d(e') \tag{4.6}$$

und sonst f_b.

Sei $\mathcal{E}_c$ die Menge der Kanten im Skelett, die noch nicht betrachtet wurden. Falls Fläche f_a als Fläche $f_{\mu,\mu'}$ gewählt wurde, werden die Werte $\delta_{\text{ext}}(\mu')$ und $\delta_{\text{ref}}(\mu')$ wie folgt gesetzt:

$$\delta_{\text{ref}}(\mu') \quad = \quad \delta_{\text{ext}}(\mu) + \sum_{e' \in \mathcal{E}_b} \ell^d(e') + \sum_{e' \in \mathcal{E}_c} \ell^d(e') \tag{4.7}$$

$$\delta_{\text{ext}}(\mu') \quad = \quad \delta_{\text{ref}}(\mu) + \sum_{e' \in \mathcal{E}_a} \ell^d(e') \tag{4.8}$$

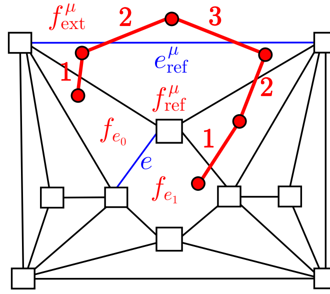


Abbildung 4.12: Wahl der Expansionsfläche in einer R-Komponente. Die Fläche f_{e_0} hat Abstand 1 zu f^{μ}_{ref} und Abstand 2 zu f^{μ}_{ext}, die Fläche f_{e_1} hat Abstand 2 zu f^{μ}_{ref} und Abstand 3 zu f^{μ}_{ext} unter der Annahme, dass die gekreuzten Kanten Länge 1 haben.

Ansonsten werden sie wie in den folgenden Gleichungen gesetzt:

$$\delta_{\text{ref}}(\mu') = \delta_{\text{ref}}(\mu) + \sum_{e' \in \mathcal{E}_a} \ell^d(e') + \sum_{e' \in \mathcal{E}_c} \ell^d(e') \qquad (4.9)$$

$$\delta_{\text{ext}}(\mu') = \delta_{\text{ext}}(\mu) + \sum_{e' \in \mathcal{E}_b} \ell^d(e') \qquad (4.10)$$

Wenn f_a als Fläche $f_{\mu,\mu'}$ gewählt wird, wird e_i zu $\mathcal{E}_a$ hinzugefügt und die nächste Kante vor e_i eingebettet, sonst wird e_i zu $\mathcal{E}_b$ hinzugefügt und die nächste Kante hinter e_i eingebettet.

R-Komponente. Es existieren genau zwei spiegelsymmetrische Einbettungen. In Abbildung 4.6 ist dargestellt, nach welchen Kriterien die Einbettung gewählt wird. Das Ziel ist es, in die Fläche $f_{\nu,\mu}$ zu expandieren, so dass in dem Beispiel die Einbettung aus Abbildung 4.6(b) gewählt würde und in $f_{\nu,\mu}$ eingebettet würde. Falls μ der Wurzelknoten von $\mathcal{T}$ ist, wird die Fläche f^{μ}_{ext} als Außenfläche gewählt, welche maximale Größe hat. Diese Fläche enthält eine reale Kante e^{μ}_{ext}. Falls μ nicht die Wurzel ist, sei e_{ext} die Referenzkante von skeleton(μ) und es wird die Fläche f^{μ}_{ext} als Außenfläche gewählt, welche e^{μ}_{ext} enthält und maximale Größe hat. Die Größe einer Fläche ist die Summe der Knotenlängen plus die Summe der Kantenlängen aller Knoten und Kanten in dieser Fläche. Die Kantenlänge für eine Kante e sei $\ell(e)$ für alle realen Kanten und gleich der Komponentenlänge (siehe Definition 3.1) für alle virtuellen Kanten. Die zu e^{μ}_{ext} adjazente Fläche ungleich f^{μ}_{ext} sei f^{μ}_{ref}.

Sei e eine virtuelle Kante mit zugehörigem Knoten μ'. Die Fläche $f_{\mu,\mu'}$ wird wie folgt gewählt (vergl. Abbildung 4.12): Seien f_{e_0} und f_{e_1} die beiden zu e adjazenten Flächen. Es wird der duale Graph der Einbettung von skeleton(μ) berechnet, wobei die Kantenlänge zwischen zwei Flächen f_i, f_j, $i \neq j$, welche mindestens eine gemeinsame Kante besitzen, gleich dem minimalen $\ell^d(e')$ über alle gemeinsamen Kanten e' ist. Mit $\pi(f_1, f_2)$ wird die Länge des kürzesten Weges zwischen f_1 und f_2 im dualen

Graphen bezeichnet. Sei $f' = f_{e_0}$ und $f'' = f_{e_1}$, falls

$$\begin{aligned} &\min\{\delta_{\text{ref}}(\mu) + \pi(f_{e_0}, f^{\mu}_{\text{ref}}); \delta_{\text{ext}}(\mu) + \pi(f_{e_0}, f^{\mu}_{\text{ext}})\} \\ \leq\ &\min\{\delta_{\text{ref}}(\mu) + \pi(f_{e_1}, f^{\mu}_{\text{ref}}); \delta_{\text{ext}}(\mu) + \pi(f_{e_1}, f^{\mu}_{\text{ext}})\}, \end{aligned} \tag{4.11}$$

und sonst $f' = f_{e_1}$ und $f'' = f_{e_0}$. Fläche f' wird als Fläche $f_{\mu,\mu'}$ gewählt. Die beiden Werte $\delta_{\text{ref}}(\mu')$ und $\delta_{\text{ext}}(\mu')$ berechnen sich wie folgt:

$$\delta_{\text{ref}}(\mu') \ = \ \delta_{\text{ref}}(\mu) + \min\{\pi(f'', f^{\mu}_{\text{ref}}); \pi(f'', f^{\mu}_{\text{ext}})\} \tag{4.12}$$

$$\delta_{\text{ext}}(\mu') \ = \ \delta_{\text{ext}}(\mu) + \min\{\pi(f', f^{\mu}_{\text{ref}}); \pi(f', f^{\mu}_{\text{ext}})\} \tag{4.13}$$

Laufzeit und Korrektheit

Die Laufzeit für eine S-Komponente ist vernachlässigbar und die Laufzeit einer R-Komponente ist aufgrund der gleichen Überlegungen wie bei der Berechnung der Dicke einer R-Komponente linear. In einer P-Komponente dominiert das Sortieren der Kanten nach ihrer Länge die Laufzeit. Im Worst-Case beinhaltet eine P-Komponente $O(|E|)$ viele Kanten. Die untere Schranke für die Worst-Case Rechenzeit für das Sortieren von m Elementen ist $\Omega(m \cdot \log(m))$ [Kro87], Merge-Sort [Knu98] beispielsweise garantiert diese Laufzeit. Im späteren Gebrauch des Algorithmus für einen allgemeinen Graphen $G' = (V', E')$ gilt, dass eine Kante maximale Länge $|V'| + |E'|$ hat — dies entspricht der maximalen Größe einer Fläche, welche alle Knoten und Kanten des Graphen enthält. Dadurch lässt sich Bucket-Sort [CL04] benutzen um alle Kanten von G in Zeit $O(m + |V'| + |E'|)$, also in linearer Zeit, nach ihrer Länge zu sortieren. Anschließend wird diese Sortierung durchlaufen und die Sortierung der Kanten einer P-Komponente übernommen. Dies sind einmalig zusätzliche Kosten $O(m)$, wodurch sich insgesamt Zeit $O(m + |V'| + |E'|)$ für das Sortieren ergibt. Die Größe von $\mathcal{T}$ samt seiner Skelettgraphen ist linear in der Größe von G, das Sortieren der Kanten in einer P-Komponente kann einmalig mit Kosten $O(m+|V'|+|E'|)$ erledigt werden und die restliche Laufzeit einer P-Komponente sowie die Laufzeit einer S- und R-Komponente ist linear. Damit ergibt sich auch insgesamt lineare Laufzeit.

Die Korrektheit folgt direkt daraus, dass eine Fläche maximiert werden kann, indem alle Kanten in diese Fläche expandiert werden (vergl. [GM04b]) und der Algorithmus eine virtuelle Kante jeweils in die adjazente Fläche expandiert, welche den kürzesten Abstand zur Außenfläche besitzt. Dadurch werden zuerst alle Flächen mit Abstand 1 zur Außenfläche — die Außenfläche selber ist gegeben und wird nicht verändert — maximiert, bei Fixierung dieser Flächen werden alle Flächen mit Abstand 2 maximiert usw. Die daraus entstehende Einbettung besitzt eine Schichtenmenge mit maximaler lexikografischer Größe. Der Algorithmus für zweizusammenhängende Graphen ist also exakt. Wie später gezeigt wird besteht das Problem darin, dass die Knotenlängen im Algorithmus für maximale Außenfläche nur die Anzahl Kanten in der Außenfläche eines Blocks speichern. Es ist also nur bekannt, wie groß Schicht $\mathcal{S}_0$ ist, aber nicht, welche Größe die anderen Schichten besitzen.

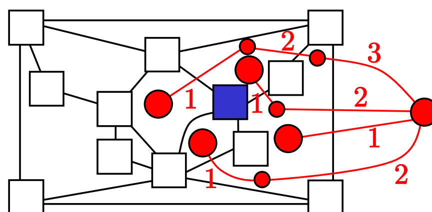


Abbildung 4.13: Abstand von Flächen adjazent zu einem Schnittknoten zur Außenfläche.

4.4.2 Zusammenhängende Graphen

Für einen Graphen $G = (V, E)$ ist eine Einbettung Γ_G mit Außenfläche f_{ext} gesucht, welche die lexikografisch größte Schichtenmenge hat. Der Algorithmus, der dieses Problem löst, berechnet zuerst den BC-Baum $\mathcal{B}$ von G (vergl. Kapitel 3.2). Als Wurzel wird ein beliebiger Block B_r gewählt, welcher eine maximale Außenfläche f_{ext} enthält. Es werden nun Knoten- und Kantenlängen definiert. Die Länge $\ell(e)$ sei für alle Kanten e gleich 1. Die Knotenlänge $\ell(v)$ sei wie in Kapitel 3.2 definiert: $\ell(v) = 0$ für alle Knoten v, die kein Schnittknoten sind und $\ell(c) = \text{smf}_{B_r}(c)$ für einen Schnittknoten c (vergl. Definition 3.2).

Es wird rekursiv die Einbettung Γ_G berechnet. Begonnen wird mit $B = B_r$ und $c = c_\emptyset$, wobei $c_\emptyset$ ein neuer Knoten ist, welcher nicht zum Graphen gehört. Für B wird mit dem Algorithmus aus Kapitel 4.4.1 eine Einbettung berechnet. Für alle Schnittknoten c', $c' \neq c$ in B, wird zuerst für alle Blöcke B', die Kinder von c' in $\mathcal{B}$ sind, rekursiv eine Einbettung berechnet (mit $c = c'$). Diese wird in die Fläche adjazent zu c' eingebettet, welche den geringsten Abstand zur Außenfläche f^B_{ext} hat (vergl. Abbildung 4.13). Algorithmisch lässt sich dies wie in einer R-Komponente für den zweizusammenhängenden Fall umsetzen. Es wird der duale Graph berechnet und ein SSSP mit der externen Fläche als Source aufgerufen. Dann werden die Blöcke in eine Fläche eingebettet, die adjazent zu dem Schnittknoten ist, welcher B und B' verbindet, und für die ein kürzester Weg zur externen Fläche existiert.

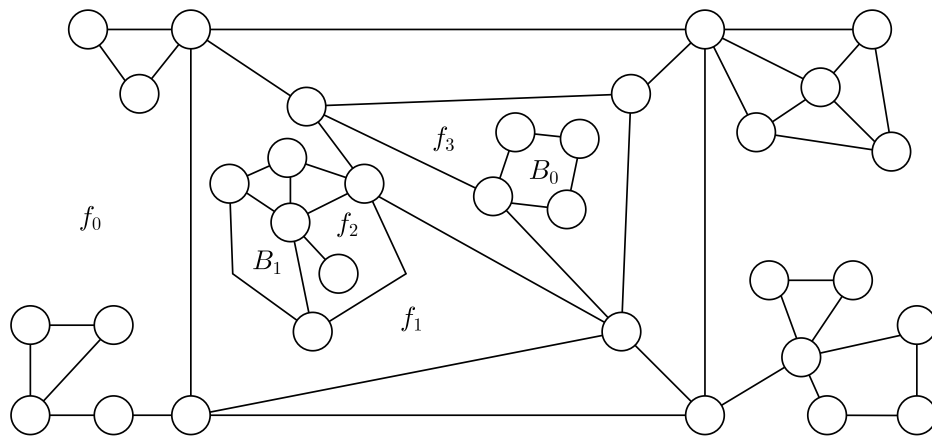
Die Größe von $\mathcal{B}$ inklusive seiner Blöcke ist linear in der Größe von G. Falls ein Block in eine innere Fläche eingebettet werden muss, kann die Entscheidung, welche der möglichen Flächen gewählt wird, nach gleichen Überlegungen wie bei einer R-Komponente, in linearer Zeit getroffen werden. Insgesamt ist die Laufzeit des Algorithmus aufgrund der linearen Laufzeit des Algorithmus für die Zweizusammenhangskomponenten linear, da die Kantenlängen in einer P-Komponente durch $O(|V| + |E|)$ beschränkt werden kann.

Der Algorithmus berechnet nicht zwangsweise die korrekte Lösung. In Abbildung 4.14 ist ein Beispiel dargestellt, indem der Algorithmus versagen kann. Die Blöcke B_0 und B_1 besitzen die gleiche Größe der Außenfläche, Schicht $\mathcal{S}_1$ hat jedoch unterschiedliche Größe. Der Algorithmus für zweizusammenhängende Graphen kennt jedoch nur die Größe der Außenfläche (durch die Knotenlänge), jedoch nicht die Größe von $\mathcal{S}_1$. Daher könnte die Einbettung wie in Abbildung 4.14(b) berechnet werden, optimal hingegen wäre die Einbettung in Abbildung 4.14(a). Würde der Algorith-

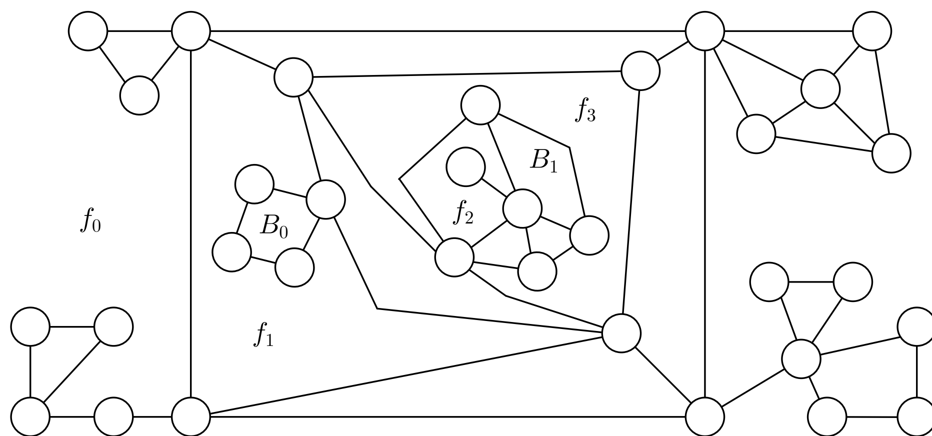
mus zur Berechnung der Knotenlängen erweitert werden, sodass die Innenkreisgröße aller Schichten in den Knotenlängen gespeichert würden, würde der Algorithmus die exakte Lösung berechnen können. Wie schon erwähnt wurde darauf aber aus Kompatibilitätsgründen vorerst verzichtet. Im Moment genügt es, um zwischen der Variante für große äußere Schichten und der Variante ohne diese Erweiterung zu wechseln, den Algorithmus zur Erstellung einer Einbettung auszutauschen. Durch Verändern der Knotenlängen wäre dies nicht mehr so einfach möglich.

Der Algorithmus lässt sich, ähnlich wie im Algorithmus für minimale Tiefe und maximale Außenfläche, folgendermaßen zu einem neuen Algorithmus erweitern. Benutzt wird das in Kapitel 3.4 vorgestellte Längenattribut (d, ℓ) mit der linearen Ordnung nach Gleichung (4.14). Dadurch wird eine planare Einbettung berechnet, in der die maximale topologische Verschachtelung eines Blocks minimal ist. Über alle Schichtenmengen mit minimaler Verschachtelung wird die lexikografisch größte Schichtenmenge in Bezug auf die Kantenanzahl berechnet. Anschaulich bedeutet dies, dass mit der Modifikation versucht wird, alle Blöcke möglichst nah an der Außenfläche einzubetten. Über alle Einbettungen, die optimal nach diesem Kriterium sind, wird die Einbettung gewählt, in welcher die Kanten möglichst nah an der Außenfläche liegen.

$$(d, \ell) > (d', \ell') \quad \Longleftrightarrow \quad d > d' \text{ oder } (d = d' \text{ und } \ell > \ell') \tag{4.14}$$



(a) optimale Lösung



(b) mögliche Lösung der Heuristik

Abbildung 4.14: Beispiel, in dem der Algorithmus unter Umständen versagt.

5 Experimentelle Vergleiche

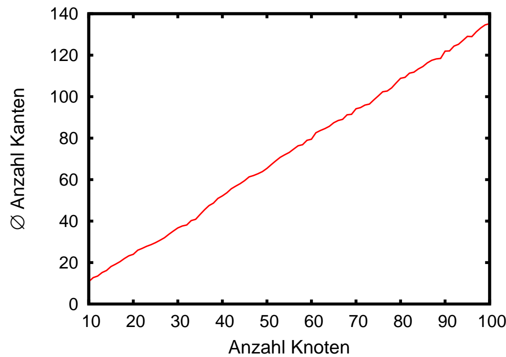
Die vorgestellten Algorithmen sollen in diesem Kapitel miteinander verglichen werden. Zunächst wird in Kapitel 5.1 der Unterschied zwischen den in Definition 2.34 und Definition 2.37 vorgestellten Tiefendefinitionen einer planaren Einbettung untersucht. Dazu werden die Algorithmen aus den Kapiteln 3.1 und 3.3 jeweils in der dort vorgestellten Version für minimale Tiefe und minimale erweiterte Tiefe gegenübergestellt. Dann werden in Kapitel 5.2 die Algorithmen aus Kapitel 3 analysiert, wobei für die Algorithmen, welche die Tiefe der Einbettung minimieren, jeweils die Variante des Algorithmus für minimale erweiterte Tiefe benutzt wurde. Die verschiedenen Varianten für die Einbettung von inneren Blöcken aus Kapitel 4 werden einzeln in Kapitel 5.3.1 untersucht. Die Varianten der Algorithmen für maximale Außenfläche sowie minimale Tiefe und maximale Außenfläche, welche dort am besten abgeschnitten haben, werden dann in Kapitel 5.3.2 gegen die restlichen untersuchten Algorithmen verglichen.

Alle Algorithmen wurden in C++ implementiert und in das Open Graph Drawing Framework (OGDF) integriert. Die Verwendung des OGDF bietet für die Diplomarbeit große Vorteile, da in ihr viele Datenstrukturen für den Umgang mit Graphen vorhanden sind, wie z.B. eine Implementierung der SPQR- und BC-Bäume. Das OGDF wird seit 2005 an der Universität Dortmund und von der oreas GmbH[1] als Open Source Projekt unter der GNU General Public Licence[2] (GPL) entwickelt.

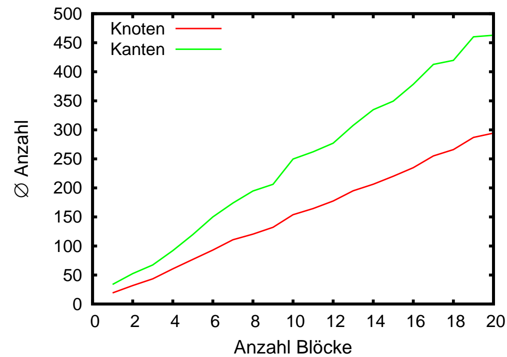
In den Experimenten wurde der Topology-Shape-Metrics Ansatz (vergl. Kapitel 2.2) angewendet. Es wurde sichergestellt, dass für alle Algorithmen zur Berechnung einer planaren Einbettung derselbe Graph als Eingabe benutzt wurde. Dazu wurden alle nicht planaren Graphen gleich planarisiert [GM04a]. Alle Schritte im Topology-Shape-Metrics Ansatz, bis auf den Algorithmus zum Berechnen einer planaren Einbettung, wurden gleich gewählt. Es wurde für alle Graphen derselbe Zeichner, die Klasse `UMLOrthoLayout` [EG+04] im OGDF, benutzt. Damit hängt die Güte einer Zeichnung von der Verwendung des Einbettungs-Algorithmus ab und die Ergebnisse können fair miteinander verglichen werden. Der Zeichner berechnet für Graphen mit maximalen Knotengrad vier eine Zeichnung mit optimaler Knickanzahl für eine fixe planare Einbettung.

Die Tests wurden mit zwei verschiedenen Testgraphensammlungen durchgeführt. Die erste Sammlung wird im weiteren Rome-Library genannt und beinhaltet 11.582 Graphen, welche von 112 reale Welt Graphen abgeleitet wurden. Dazu wurden Graphen aus Anwendungen, wie z.B. Datenbanken, genommen und diese Graphen durch Hinzufügen von Knoten und Kanten variiert, um so aus wenigen Graphen eine große-

[1] `http://www.oreas.com/`, Stand: 16. August 2007
[2] `http://www.gnu.org/licenses/gpl-2.0.html`, Stand: 16. August 2007



(a) Durchschnittliche Anzahl Kanten der Graphen in der Rome-Library.



(b) Eigenschaften der Graphen in der Block-Library.

Abbildung 5.1: Eigenschaften der Testgraphensammlungen.

re Menge Testgraphen zu erzeugen (siehe [BG+97] für Details). Die Graphensammlung kann unter [GDT] runtergeladen werden. Abbildung 5.1(a) zeigt die durchschnittliche Anzahl der Kanten.

Die zweite Graphensammlung wurde erzeugt, um die Algorithmen für Graphen mit bestimmten Eigenschaften zu testen, und wird im folgenden Block-Library genannt. Es wurden dazu jeweils 100 planare Graphen mit einer Blockanzahl von 1 bis 20 erzeugt. Jeder Block hat eine zufällige Knotenanzahl zwischen 1 und 30 und eine zufällige Kantenanzahl zwischen n und $3n - 6$ für einen Block mit n Knoten. Abbildung 5.1(b) zeigt die durchschnittliche Anzahl an Knoten und Kanten für die Graphen der Block-Library.

Die Laufzeitmessungen wurden auf einem IBM ThinkPad T42 mit einem Intel Pentium M Prozessor mit 1.4 Ghz und 64 KB L1 sowie 1 MB L2 Cache und 1 GB RAM durchgeführt. Das OGDF wurde in einem Microsoft Visual Studio 2005 Projekt eingebunden und mit dem Visual C++ Compiler compiliert. Das Projekt befindet sich auf der beiliegenden DVD, siehe Anhang A.

Die berechneten Zeichnungen der Graphen werden anhand der folgenden Werte verglichen:

- Bends(x): Anzahl der Kantenknicke in der berechneten Zeichnung mit dem Einbettungs-Algorithmus x.

- EdgeLength(x): Summe der euklidischen Kantenlänge aller Kanten in der Zeichnung für den Einbettungs-Algorithmus x.

- Area(x): Benötigte Zeichenfläche (Größe der Bounding-Box[3]).

- InnerFaces(x): Größe der inneren Flächen, das heißt alle die Größe aller Flächen ungleich der Außenfläche.

[3] kleinstes Rechteck, das alle Objekte enthält

(a) Zeichnung mit kleinen inneren Flächen. (b) Zeichnung mit großen inneren Flächen.

Abbildung 5.2: Beispiel für zwei Zeichnungen mit unterschiedlicher Größe der inneren Flächen.

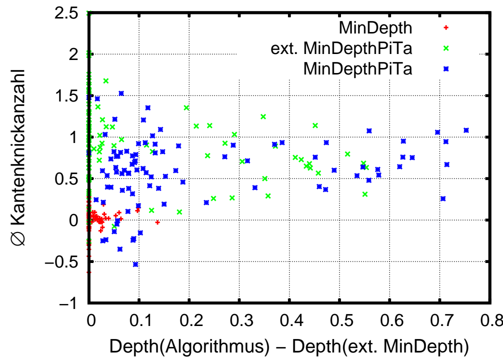
- Depth(x): Erweiterte Tiefe der planaren Einbettung berechnet mit Algorithmus x.
- Kantenanzahl in der Außenfläche der planaren Einbettung.

Die Größe der inneren Flächen ist ein Bewertungskriterium, welches in anderen Vergleichen, wie z.B. in [Piz05], noch nicht benutzt wurde. Die Idee dabei ist, dass die Einbettung eines simplen Blocks in die Außenfläche die Größe der benötigten Zeichenfläche nicht unbedingt verkleinert, aber die Größe der inneren Flächen. Vergleicht man Zeichnungen, in denen die Summe der inneren Flächen deutlich unterschiedlich ist, wie z.B. in Abbildung 5.2, die Größe der benötigten Zeichenfläche jedoch gleich, so erkennt man den ästhetischen Vorteil von kleinen inneren Flächen. Die erweiterte Tiefe der planaren Einbettung und die Anzahl der Kanten in der Außenfläche dienen nicht als Bewertungskriterien, sondern sollen Aufschluss über die Topologie der Einbettungen geben.

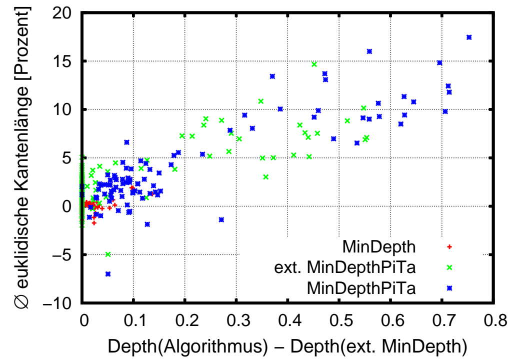
5.1 Minimale Tiefe und erweiterte minimale Tiefe

Die Motivation für die Einführung der erweiterten Tiefe ist die leichtere Verständlichkeit der Topologie eines Graphen, falls die maximale Verschachtelung von allen Blöcken in einer Zeichnung minimal ist — eben auch die von Blöcken, welche nur aus einer einzigen Kante bestehen. In diesem Kapitel wird untersucht, ob sich durch die Berechnung einer planaren Einbettung mit minimaler erweiterter Tiefe ein Vorteil gegenüber einer planaren Einbettung mit minimaler Tiefe ergibt. Weiterhin soll untersucht werden, ob es einen gravierenden Unterschied in den Ergebnissen für die Algorithmen von Pizzonia und Tamassia aus Kapitel 3.1 sowie von Gutwenger und Mutzel aus Kapitel 3.3 gibt. Zur Erinnerung: In dem Algorithmus von Pizzonia und Tamassia muss die Einbettung der Blöcke des Graphen gegeben sein, wohingegen der Algorithmus von Gutwenger und Mutzel keine Einschränkung besitzt.

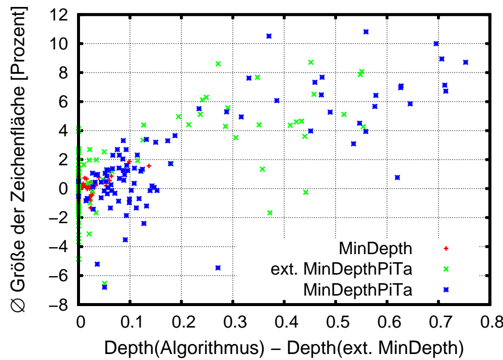
Dazu werden diese beiden Algorithmen in beiden Varianten für minimale Tiefe und für minimale erweiterte Tiefe analysiert. Zu beachten ist, dass der Algorithmus von Pizzonia und Tamassia, so wie er in Kapitel 3.1 vorgestellt wurde, die minimale Tiefe berechnet, der von Gutwenger und Mutzel minimale erweiterte Tiefe. Es wurde



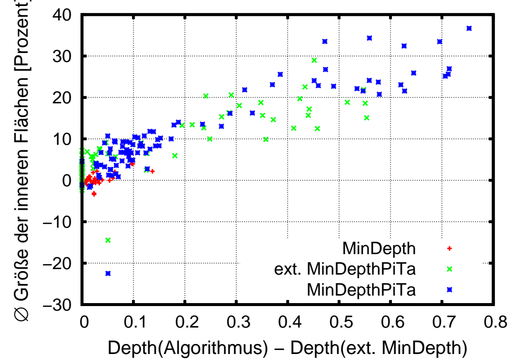
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Algorithmus) − Bends(ext. MinDepth).



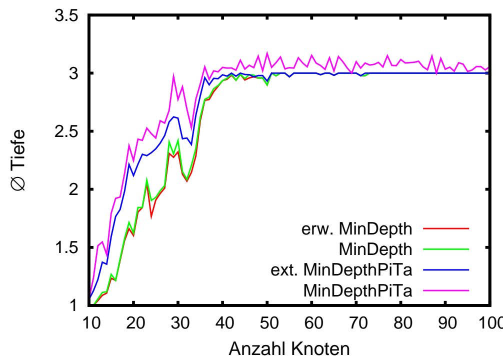
(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(ext. MinDepth) besser als EdgeLength(Algorithmus), in Prozent.



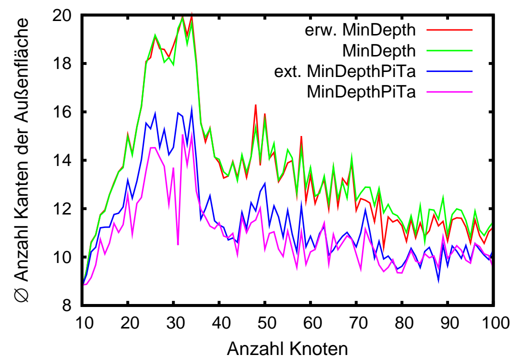
(c) Durchschnittliche Größe der Zeichenfläche: Area(ext. MinDepth) besser als Area(Algorithmus), in Prozent.



(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(ext. MinDepth) besser als InnerFaces(Algorithmus), in Prozent.



(e) Durchschnittliche erweiterte Tiefe der berechneten planaren Einbettungen.



(f) Durchschnittliche Größe der Außenfläche.

Abbildung 5.3: Experimenteller Vergleich der verschiedenen Tiefendefinitionen für Graphen aus der Rome-Library.

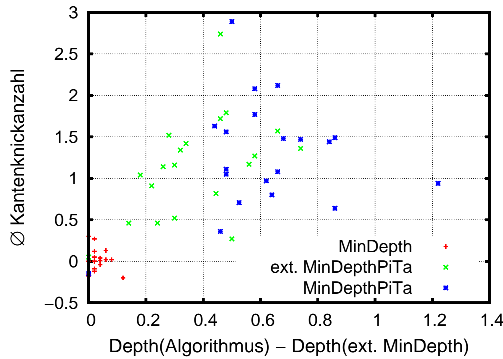
jedoch in den Kapiteln 3.1.2 und 3.3.1 beschrieben, wie sie sich für die jeweils andere Variante modifizieren lassen. Für die Pizzonia und Tamassia Algorithmen wurde die Einbettung der Blöcke mit der Funktion `planarEmbed` der Klasse `PlanarModule` im OGDF berechnet. Diese implementiert den Algorithmus aus [CN+85]. Es werden die folgenden Bezeichnung benutzt:

- *ext. MinDepth* für den minimale Tiefe Algorithmus von Gutwenger und Mutzel in der Variante für minimale erweiterte Tiefe,
- *MinDepth* für die Variante minimale Tiefe,
- *ext. MinDepthPiTa* für den Algorithmus von Pizzonia und Tamassia mit Variante minimale erweiterte Tiefe und
- *MinDepthPiTa* für die Variante minimale Tiefe.

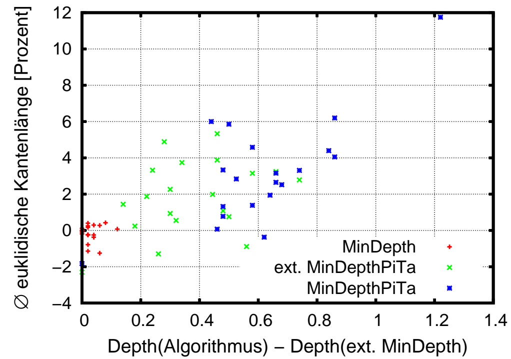
Die Abbildungen 5.3(e) und 5.4(e) zeigen die durchschnittliche erweiterte Tiefe der berechneten planaren Einbettungen. Es ist zu erkennen, dass der Unterschied zwischen ext. MinDepth und MinDepth eher gering ist, die beiden PiTa Algorithmen jedoch grundsätzlich eine Einbettung mit größerer erweiterter Tiefe berechnen. Dabei ist die erweiterte Tiefe bei MinDepthPiTa nochmal ein bisschen größer als bei ext. MinDepthPiTa. Ab ca. 50 Knoten stagniert bei den Graphen der Rome-Library die minimale erweiterte Tiefe bei 3, und alle Algorithmen, außer MinDepthPiTa, erreichen diese Tiefe auch. In die Graphen der Rome-Library werden mit steigender Knotenanzahl immer mehr Dummy-Knoten hinzugefügt, um sie zu planarisieren. Dies führt allerdings dazu, dass diese Graphen sehr dicht werden und damit die Anzahl unterschiedlicher Einbettungen immer weniger wird, weshalb sich die minimale Tiefe nicht mehr verändert.

Die Ergebnisse der Experimente in den Abbildungen 5.3 und 5.4 sind so dargestellt, dass der Einfluss der Tiefe auf das Ergebnis hervorgehoben wird. Dazu wurde als x-Achse der Unterschied der erweiterten Tiefe der berechneten planaren Einbettung mit der minimalen erweiterten Tiefe (berechnet vom Algorithmus ext. MinDepth) gewählt. Als y-Achse wurde dann der Unterschied der Güte der Lösung im Vergleich mit dem Ergebnis vom Algorithmus ext. MinDepth gewählt. Das bedeutet, dass, je mehr das Ergebnis über dem y-Wert 0 liegt, umso schlechter ist es im Vergleich zum Algorithmus ext. MinDepth.

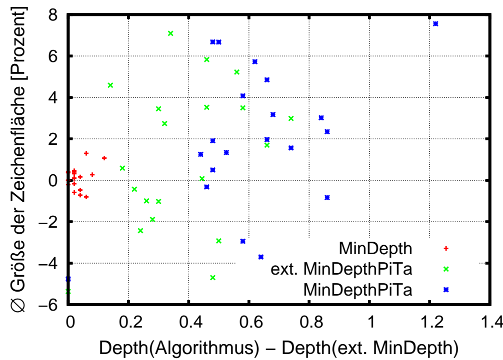
Für die Kantenknickanzahl lässt sich sowohl in der Rome-Library als auch in der Block-Library erkennen, dass das Ergebnis, je mehr die erweiterte Tiefe von der minimalen Tiefe abweicht, umso schlechter wird. Dabei sind die Ergebnisse für ext. MinDepth und MinDepth jedoch beinahe identisch, jedoch ext. MinDepthPiTa und MinDepthPiTa liefern im Durchschnitt schlechtere Ergebnisse. Für die Summe der Kantenlängen ist das Ergebnis ähnlich zu der Kantenknickanzahl. Bei der durchschnittlichen Größe der benötigten Zeichenfläche gibt es auch bei größerem Unterschied zwischen der Tiefe der berechneten Einbettung und der minimalen Tiefe Ausreißer, bei denen das Ergebnis von ext. MinDepthPiTa oder MinDepthPiTa besser als das von ext. MinDepth ist. Im Durchschnitt ist das Ergebnis bei der



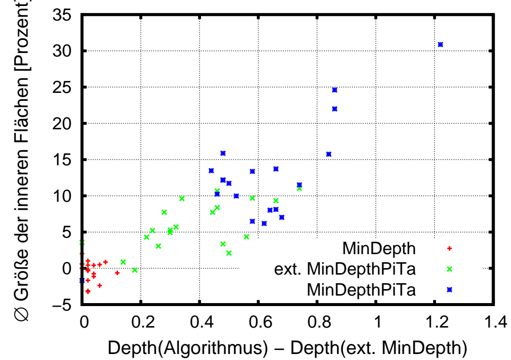
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Algorithmus) − Bends(ext. MinDepth).



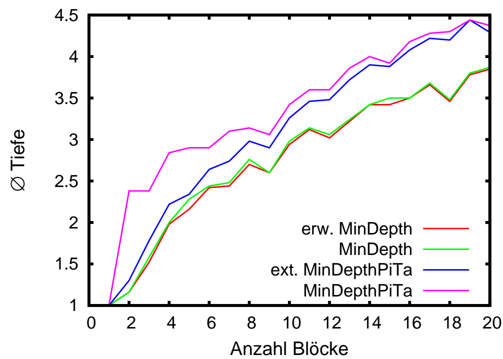
(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(ext. MinDepth) besser als EdgeLength(Algorithmus), in Prozent.



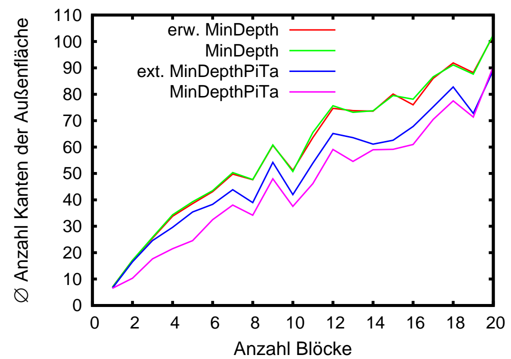
(c) Durchschnittliche Größe der Zeichenfläche: Area(ext. MinDepth) besser als Area(Algorithmus), in Prozent.



(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(ext. MinDepth) besser als InnerFaces(Algorithmus), in Prozent.



(e) Durchschnittliche erweiterte Tiefe der berechneten planaren Einbettungen.



(f) Durchschnittliche Größe der Außenfläche.

Abbildung 5.4: Experimenteller Vergleich der verschiedenen Tiefendefinitionen für Graphen aus der Block-Library.

Rome-Library jedoch eindeutig schlechter. Bei der Block-Library hingegen ist es im Durchschnitt nur minimal schlechter.

Die durchschnittliche Summe der Größe der inneren Flächen zeigt bei den Graphen der Rome-Library einen deutlichen Vorteil von ext. MinDepth gegenüber den anderen Algorithmen. Bei der Block-Library ist zwar ein Vorteil gegenüber den Algorithmen ext. MinDepthPiTa und MinDepthPiTa zu erkennen, im Durchschnitt ist MinDepth aber ein wenig besser als ext. MinDepth. Es wurde damit bestätigt, dass es sinnvoll ist, nicht nur die Größe der insgesamt benötigten Zeichenfläche zu betrachten, sondern auch die Größe der inneren Flächen. Bei den benötigten Zeichenfläche ist das Ergebnis eher durchwachsen, bei der Größe der inneren Flächen hingegen ist ein eindeutiges Ergebnis zu erkennen, welches für Zeichnungen mit minimaler erweiterter Tiefe spricht.

Alle Ergebnisse betrachtend bleibt festzuhalten, dass der Algorithmus für minimale Tiefe von Gutwenger und Mutzel bessere Ergebnisse liefert als der von Pizzonia und Tamassia. Weiterhin zeigen sich Vorteile, wenn auch eher geringe, für die minimale erweiterte Tiefe im Vergleich mit der minimalen Tiefe.

5.2 Algorithmen zur Berechnung von planaren Einbettungen

Die in Kapitel 3 vorgestellten Algorithmen werden in diesem Kapitel analysiert. Es werden dabei die folgenden Bezeichnungen für die Algorithmen benutzt:

- *Simple*: Erzeugt eine planare Einbettung mit der Funktion `planarEmbed` der Klasse `PlanarModule` im OGDF. Die Funktion ist eine Implementierung des Algorithmus aus [CN+85]. Als Außenfläche wird die Fläche mit maximaler Größe (maximale Kantenanzahl) gewählt.

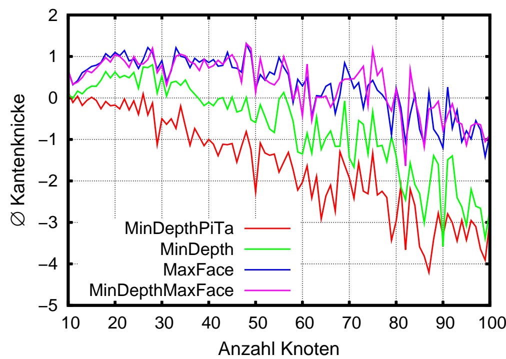
- *MinDepthPiTa*: Algorithmus für minimale Tiefe von Pizzonia und Tamassia (Kapitel 3.1) in der Variante für minimale erweiterte Tiefe. Die Einbettung der Blöcke wurde wieder mit der Funktion `planarEmbed` berechnet.

- *MinDepth*: Algorithmus für minimale erweiterte Tiefe (Kapitel 3.3).

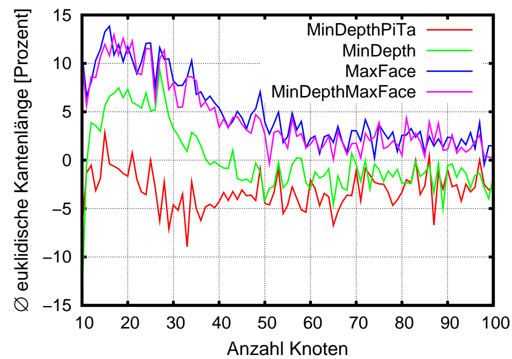
- *MaxFace*: Algorithmus für maximale Außenfläche (Kapitel 3.2).

- *MinDepthMaxFace*: Algorithmus für minimale Tiefe und maximale Außenfläche (Kapitel 3.4). Es wurde dabei die minimale erweiterte Tiefe berechnet.

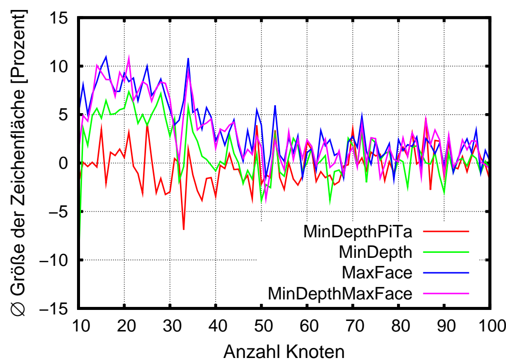
Abbildung 5.5 stellt die Ergebnisse der Tests für die Graphen der Rome-Library dar. Eine generelle Erkenntnis für alle Bewertungskriterien ist die, dass MinDepthPiTa der schlechteste Algorithmus ist. Am zweit schlechtesten schneidet MinDepth ab und die beiden Algorithmen MaxFace und MinDepthMaxFace sind die Sieger dieses Vergleichs. Eine Besonderheit, die auffällt, ist, dass für die Kantenknickanzahl ab 80 Knoten alle Algorithmen schlechter abschneiden als Simple. In die Graphen der



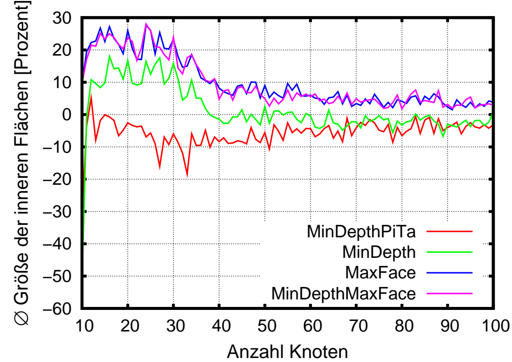
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



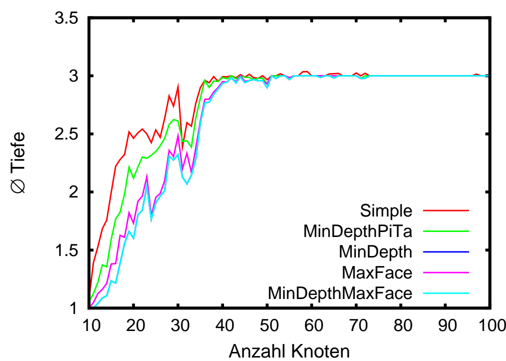
(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.



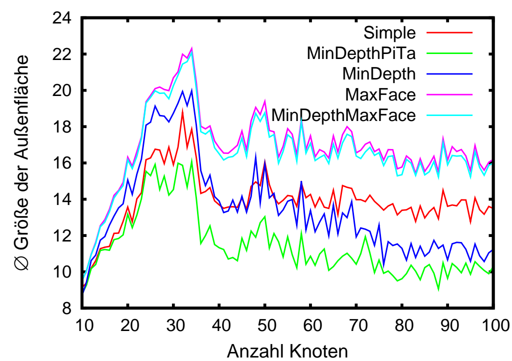
(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.



(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.



(e) Durchschnittliche erweiterte Tiefe der berechneten planaren Einbettungen.



(f) Durchschnittliche Größe der Außenfläche.

Abbildung 5.5: Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Rome-Library.

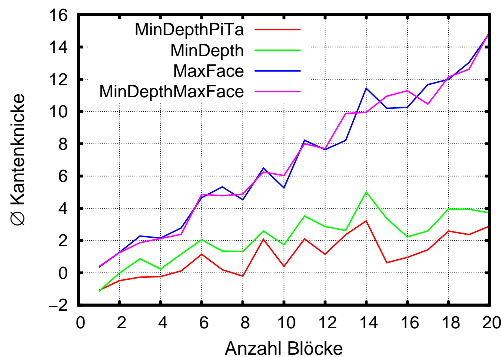
Rome-Library werden mit steigender Knotenanzahl immer mehr Dummy-Knoten hinzugefügt, um sie zu planarisieren. Dies führt allerdings dazu, dass diese Graphen sehr dicht werden und damit die Anzahl unterschiedlicher Einbettungen immer weniger wird und auch das Optimierungspotential sinkt. Dies gilt nicht nur für die Kantenknickanzahl, sondern auch für die anderen Bewertungskriterien.

Für alle übrigen Kriterien sind aber, zumindest die Sieger, immer besser als Simple. Dies lässt vermuten, dass eine kantenknickminimale Zeichnung nicht immer optimal ist. Es kann also von Vorteil sein, einen Kantenknick mehr zu „investieren“, wenn damit beispielsweise die Länge von einigen Kanten erheblich verringert werden kann.

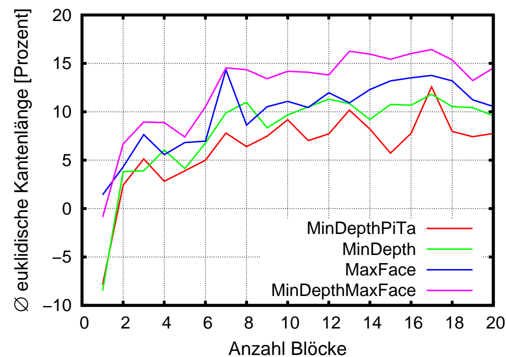
Die Abbildungen 5.5(e) und 5.5(f) zeigen die durchschnittliche erweiterte Tiefe der berechneten planaren Einbettungen und die durchschnittliche Anzahl Kanten in der Außenfläche. Das Ergebnis für MinDepth und MinDepthMaxFace sind für die erweiterte Tiefe natürlich identisch. Interessant ist, dass MaxFace immer eine Einbettung mit Tiefe nahe der minimalen Tiefe erzeugt und MinDepthMaxFace eine Einbettung mit einer Größe der Außenfläche, welche nicht viel kleiner ist als die maximal mögliche Größe der Außenfläche, das heißt, die beiden Sieger ähneln sich in diesen beiden Eigenschaften. MinDepth berechnet natürlich eine Einbettung mit minimaler Tiefe, die Größe der Außenfläche ist aber im Durchschnitt deutlich kleiner als die maximale Größe. Wie schon erwähnt sind die Ergebnisse von MinDepth schlechter als die von MaxFace und MinDepthMaxFace, was darauf hinweist, dass die Größe der Außenfläche ein wichtiger Parameter für die Güte der Zeichnung ist. Die Tiefe der Einbettungen stagniert für alle Algorithmen schon ab 60 Knoten bei ca. 3, welches wie oben beschrieben auf die steigende Dichte zurückzuführen ist.

In Abbildung 5.6 sind die Ergebnisse für die Block-Library dargestellt. Bei der Anzahl der Kantenknicke sind MaxFace und MinDepthMaxFace wieder die besten Algorithmen, je mehr Blöcke der Graph enthält umso deutlicher wird der Unterschied zu den anderen. MinDepth ist zwar etwas besser als MinDepthPiTa, aber beide sind nur geringfügig besser als Simple. Bei der durchschnittlichen Summe der Kantenlängen ist MinDepthMaxFace am besten. MaxFace ist hier etwas schlechter als MinDepthMaxFace und nur etwas besser als MinDepth. Generell liegen die Ergebnisse recht nah beieinander, auch wenn alle Algorithmen besser als Simple abschneiden — zumindest ab einer Blockanzahl von zwei. MinDepthMaxFace siegt auch bei der Größe der benötigten Zeichenfläche. Erstaunlicherweise schneidet hier MinDepth allerdings teilweise als bestes ab und kann gut mit MaxFace mithalten. Die Ergebnisse sind jedoch nicht sehr eindeutig und liegen nah zusammen. Betrachtet man hingegen die Größe der inneren Flächen, ergibt sich ein deutlicherer Unterschied. Die Algorithmen MinDepthMaxFace und MaxFace schneiden am besten ab, mit einem kleinem Vorsprung für MinDepthMaxFace.

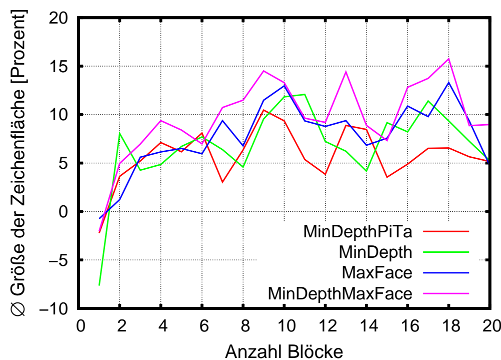
Betrachtet man hier die Tiefe der Einbettungen, so ist der Unterschied zwischen MinDepth bzw. MinDepthMaxFace und MaxFace größer als bei den Rome-Graphen. MinDepthPiTa ist sehr ähnlich zu MaxFace, die Tiefe der Einbettungen, die Simple berechnet, ist nochmal einiges größer als die von MinDepthPiTa und MaxFace. Auch der Unterschied bei der Größe der Außenfläche zwischen MaxFace und MinDepth-



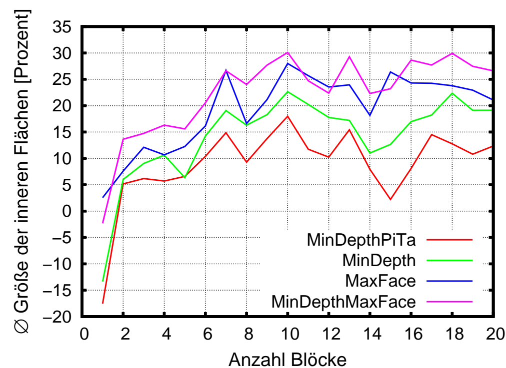
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



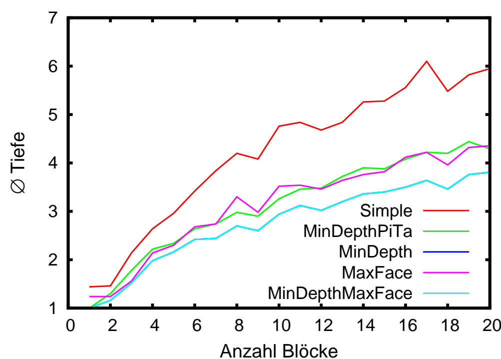
(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.



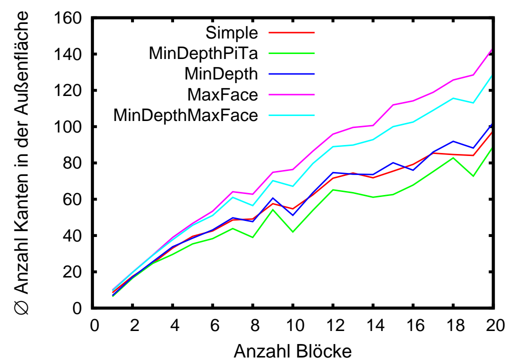
(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.



(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.



(e) Durchschnittliche erweiterte Tiefe der berechneten planaren Einbettungen.



(f) Durchschnittliche Größe der Außenfläche.

Abbildung 5.6: Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Block-Library.

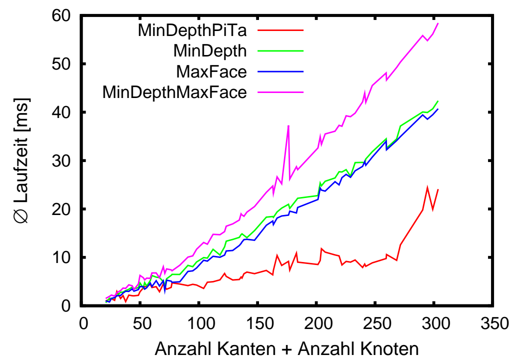


Abbildung 5.7: Messung der Laufzeit der Algorithmen.

MaxFace ist größer als bei den Rome-Graphen. Da in diesen Tests MinDepthMaxFace aber besser als MaxFace abgeschnitten hat, scheint es wichtig zu sein, nicht nur die Außenfläche zu maximieren, sondern auch die Tiefe zu minimieren. Wie allerdings schon vorher gesehen reicht das Minimieren der Tiefe alleine nicht aus.

Die Laufzeit der Algorithmen ist in Abbildung 5.7 dargestellt. Die Laufzeit bezieht sich dabei nur auf die Berechnung der planaren Einbettung und nicht auf die Gesamtzeit, die benötigt wurde, um die Zeichnung zu erstellen. Die geringste Laufzeit benötigt MinDepthPiTa. Die Laufzeit ist allerdings davon abhängig, ob die im ersten Schritt des Algorithmus berechnete Einbettung im zweiten Schritt modifiziert werden muss, oder ob der Schritt übersprungen werden kann. Dies erklärt, warum ab einer Summe der Knoten und Kanten ab ca. 275 die Laufzeit stark wächst, da diese Graphen eine Topologie besitzen, in der eine „Korrektur“ der ersten erstellten Einbettung öfter benötigt wird. Die Algorithmen MinDepth und MaxFace benötigen in etwa die gleiche Zeit. Da sie algorithmisch sehr ähnlich sind, ist dies auch nicht erstaunlich. Die Laufzeit von MinDepthMaxFace ist, wie zu erwarten, am höchsten. Aufgrund der geschickten Berechnung der Einbettung ist sie aber nicht sehr viel größer als die von MinDepth und MaxFace, obwohl der Algorithmus quasi beide Parameter optimiert.

Fasst man alle Ergebnisse zusammen, dann ergeben sich zwei Sieger: MaxFace und MinDepthMaxFace. Bei den Graphen aus der Rome-Library schneiden beide gleich gut ab. Bei der Block-Library hingegen siegt MinDepthMaxFace. Das bedeutet, dass bei Graphen mit vielen Blöcken MinDepthMaxFace zu bevorzugen ist, sonst sollte man aufgrund der geringeren Rechenzeit MaxFace den Vorzug geben.

5.3 Methoden für die Einbettung von inneren Blöcken

In Kapitel 5.3.1 werden zunächst die Algorithmen untereinander getestet. Dabei werden die Varianten aus Kapitel 4 jeweils mit dem Algorithmus für maximale Außenfläche und für den Algorithmus für minimale Tiefe und maximale Außenfläche

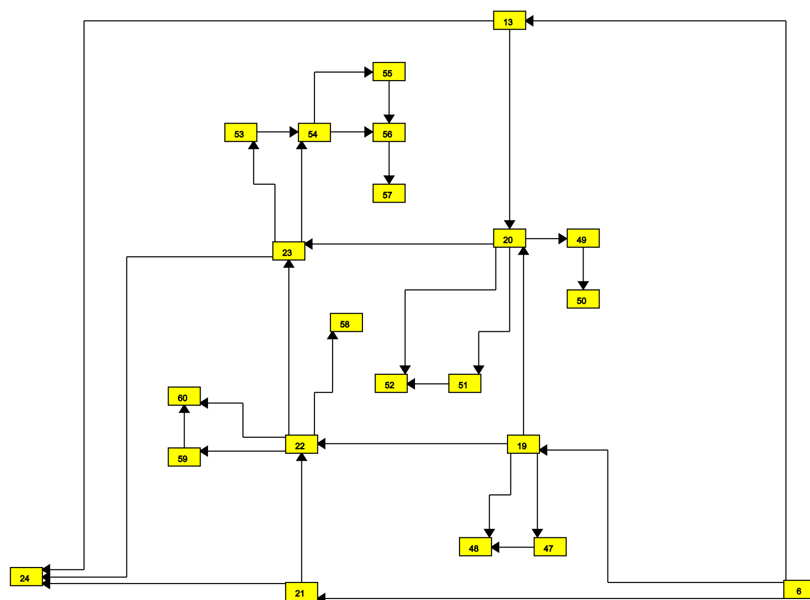


Abbildung 5.8: Planare Einbettung für G erstellt mit kleine Flächen Variante.

kombiniert und miteinander verglichen, da diese Algorithmen in den Vergleichen aus dem letzten Kapitel am besten abgeschnitten haben. Die Variante, die jeweils das beste Ergebnis liefert, wird dann in Kapitel 5.3.2 anstelle der simplen Einbettungs-Strategie für innere Blöcke mit dem Algorithmus für maximale Außenfläche und für den Algorithmus für minimale Tiefe und maximale Außenfläche kombiniert und damit die Vergleiche aus Kapitel 5.2 erneut durchgeführt.

5.3.1 Varianten im Vergleich untereinander

Es werden diese Bezeichnungen benutzt:

- *MaxFace*: Algorithmus für maximale Außenfläche mit simplem Ansatz für innere Flächen (Kapitel 4.1).

- *MaxFaceSmallFaces*: Algorithmus für maximale Außenfläche mit der Strategie, alle inneren Flächen so klein wie möglich zu halten (Kapitel 4.2). Es wurde dabei die in Algorithmus 4.1 vorgestellte Heuristik verwendet.

- *MaxFaceBigFaces*: Algorithmus für maximale Außenfläche mit der Strategie, Blöcke, die nicht in die Außenfläche eingebettet werden können, auf wenige Flächen zu verteilen (Kapitel 4.3). Es wurde dabei die in Algorithmus 4.2 vorgestellte Heuristik verwendet.

- *MaxFaceLayers*: Algorithmus für maximale Außenfläche mit der Strategie, Blöcke, die nicht in die Außenfläche eingebettet werden können, in eine Fläche möglichst nah an der Außenfläche einzubetten (Kapitel 4.4).

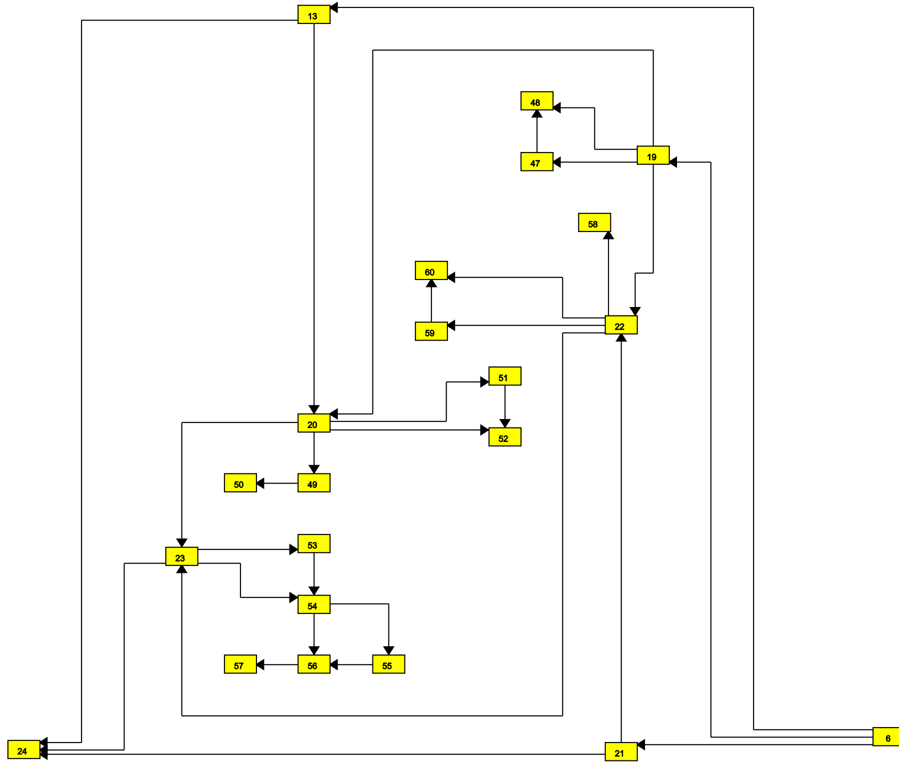


Abbildung 5.9: Planare Einbettung für G erstellt mit wenige große Flächen Variante.

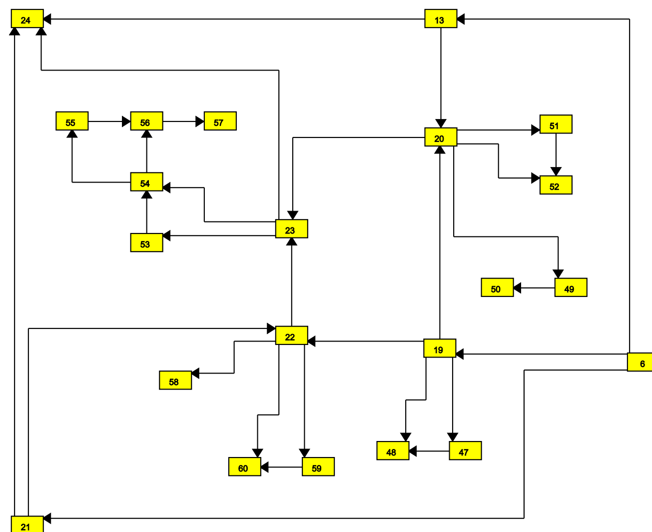


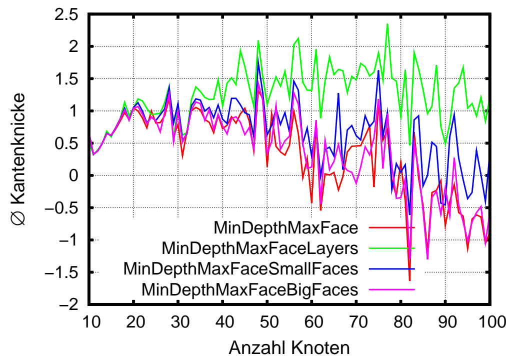
Abbildung 5.10: Planare Einbettung für G erstellt mit große äußere Schichten Variante.

- *MinDepthMaxFace*: Algorithmus für minimale Tiefe und maximale Außenfläche mit simplen Ansatz für innere Flächen.
- *MinDepthMaxFaceSmallFaces*: Algorithmus für minimale Tiefe und maximale Außenfläche mit der Heuristik für kleine inneren Flächen.
- *MinDepthMaxFaceBigFaces*: Algorithmus für minimale Tiefe und maximale Außenfläche mit der Heuristik für die Verteilung von Blöcke, die nicht in die Außenfläche eingebettet werden können, auf wenige Flächen.
- *MinDepthMaxFaceLayers*: Algorithmus für minimale Tiefe und maximale Außenfläche mit der Strategie, Blöcke, die nicht in die Außenfläche eingebettet werden können, in eine Fläche möglichst nah an der Außenfläche einzubetten.

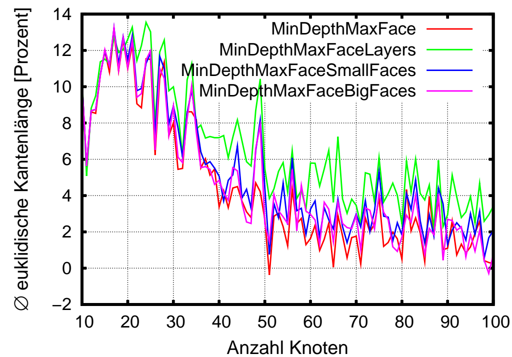
Die Abbildungen 5.8, 5.9 und 5.10 zeigen das Ergebnis für denselben Graphen G unter Verwendung der verschiedenen vorgestellten neuen Einbettungsvarianten der inneren Blöcke. Dabei wurde die Außenfläche fixiert, indem an die Knoten 24, 13, 6 und 21 Blöcke angehängt wurden, so dass die Außenfläche nur mit diesen Blöcken maximale Größe hat. Der Übersichtlichkeit halber wurden diese Blöcke jedoch nicht mit dargestellt. Es ist gut zu erkennen, wie die Algorithmen die inneren Blöcke einmal alle in dieselbe Fläche, dann auf möglichst viele Flächen verteilt und einmal in die Flächen am kürzesten entfernt von der Außenfläche einbetten.

Die Abbildungen 5.12 und 5.13 stellen die Einbettung des gleichen Blocks B dar. In dem Graph, aus dem B stammt, liegen die Knoten 0, 1, 2 und 3 in der Außenfläche. Dies wurde auch hier sichergestellt, indem an sie ein Block mit einer großen Anzahl Kanten in der Außenfläche angehängt wurde. Diese Blöcke wurden wieder nicht mit gezeichnet. Bis auf die Knoten 0, 1, 2 und 3 sind alle anderen Knoten des Graphen keine Schnittknoten. Die planare Einbettung in Abbildung 5.12 wurde mit MaxFace berechnet, da die Algorithmen MaxFaceSmallFaces und MaxFaceBigFaces die Einbettung der Zweizusammenhangskomponenten nicht verändern, entspricht die berechnete Einbettung auch dem Ergebnis bei Benutzung dieser Varianten für innere Blöcke. Die Abbildung 5.13 stellt das Ergebnis dar, welches der Algorithmus MaxFaceLayers berechnet. Vergleicht man die beiden Einbettungen miteinander, zeigt sich ein besseres Ergebnis für MaxFaceLayers. Die Anzahl der Kantenknicke ist mit 46 im Vergleich zu 50 geringer und die Summe der Kantenlängen ist um ca. 14 Prozent geringer.

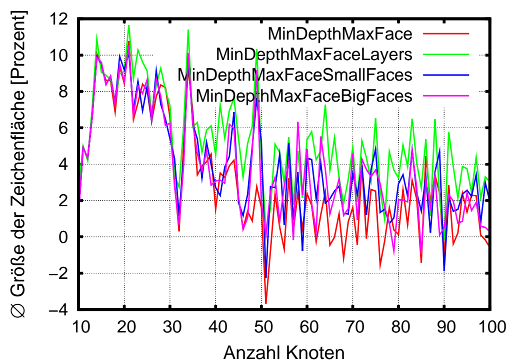
Das Ergebnis der Experimente ist eindeutig. Sowohl für die Graphen der Rome-Library, als auch für die Graphen der Block-Library, schneidet die große äußere Schichten Variante bei allen Bewertungskriterien am besten ab. Dies ist sowohl für MaxFaceLayers (vergl. Abbildungen 5.11 und 5.14) als auch für MinDepthMaxFaceLayers (vergl. Abbildungen 5.15 und 5.16) der Fall. Die SmallFaces Varianten schneiden schlechter ab als die Layers Varianten, aber leicht besser als die beiden anderen Varianten. Die Simple und BigFaces Varianten sind ähnlich gut, allerdings mit geringfügigem Vorteil für MinDepthMaxFaceBigFaces bzw. MaxFaceBigFaces. Eine kleine Ausnahme ist bei der Größe der Zeichenfläche und der Größe der inneren



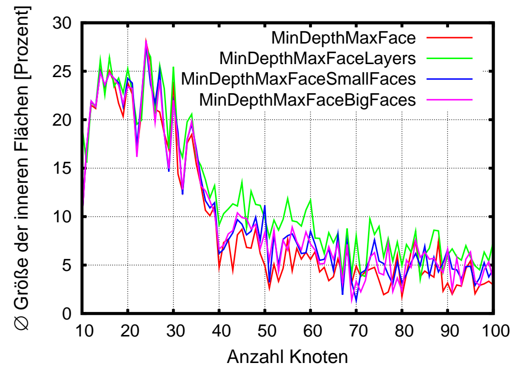
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.



(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.



(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.

Abbildung 5.11: Vergleich der Varianten aus Kapitel 4 für Graphen aus der Rome-Library mit dem Algorithmus für minimale Tiefe und maximale Außenfläche.

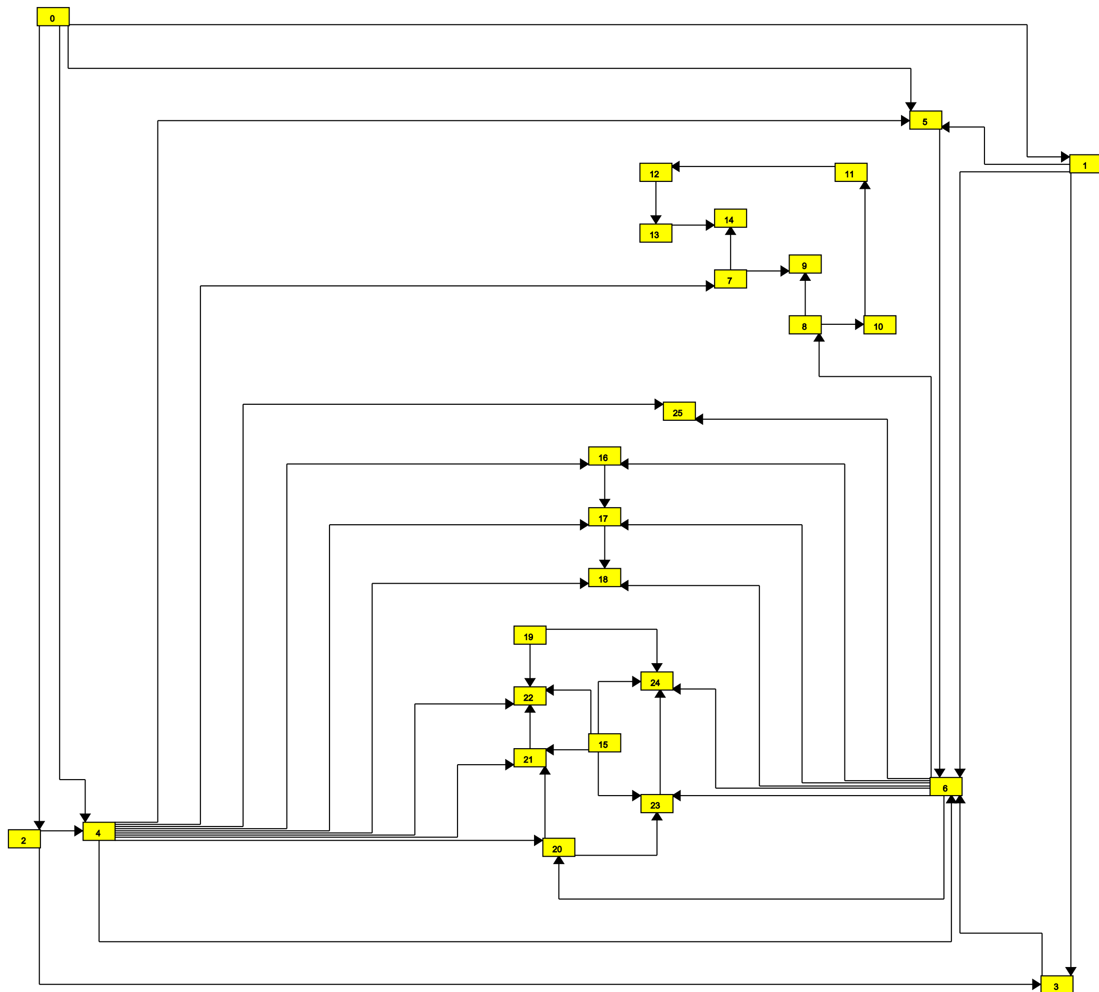


Abbildung 5.12: Einbettung von Block B berechnet mit Algorithmus für maximale Außenfläche.

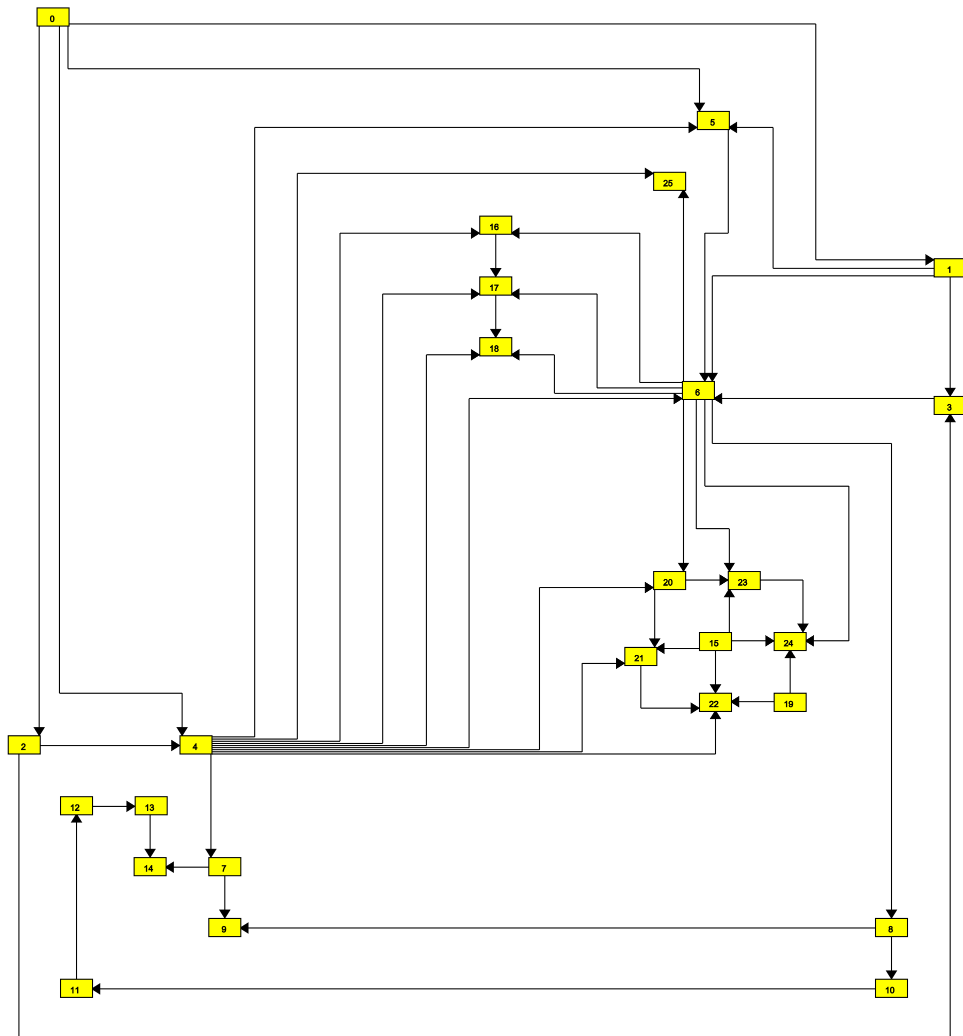
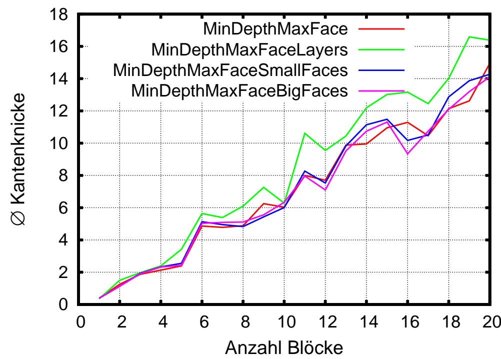
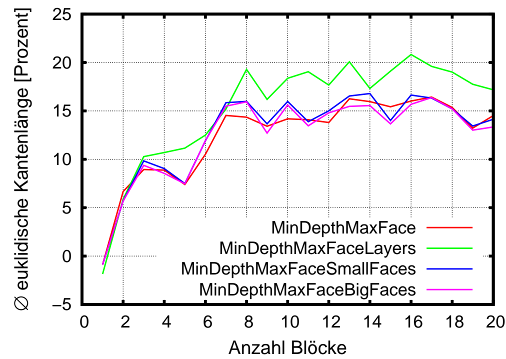


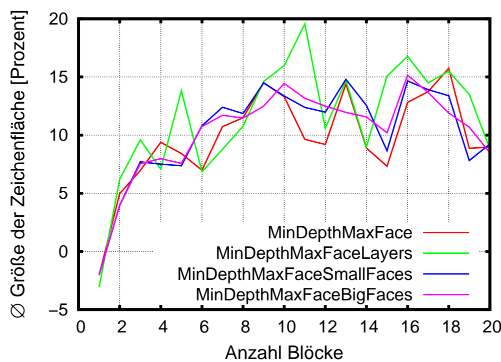
Abbildung 5.13: Einbettung von Block B berechnet mit Algorithmus für maximale Außenfläche mit großen äußeren Schichten.



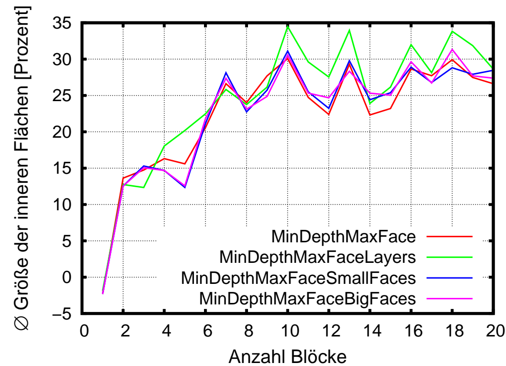
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.

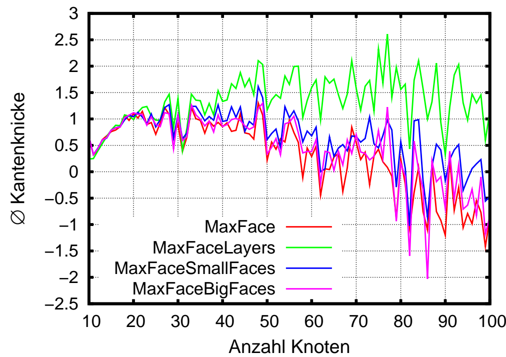


(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.

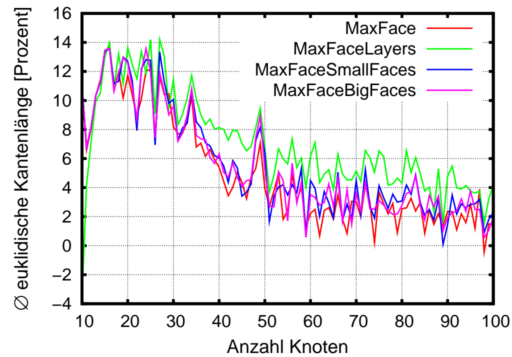


(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.

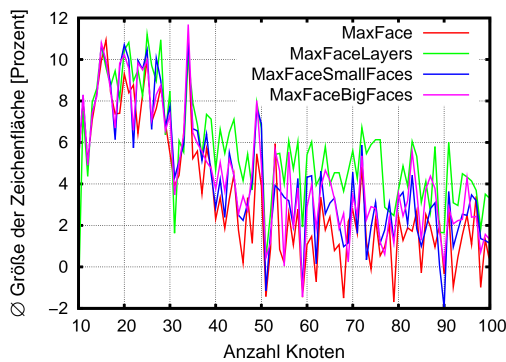
Abbildung 5.14: Vergleich der Varianten aus Kapitel 4 für Graphen aus der Block-Library mit dem Algorithmus für minimale Tiefe und maximale Außenfläche.



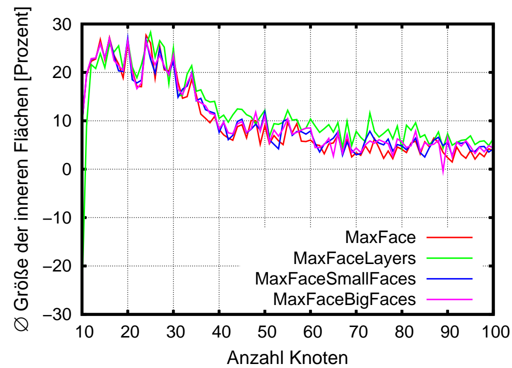
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.

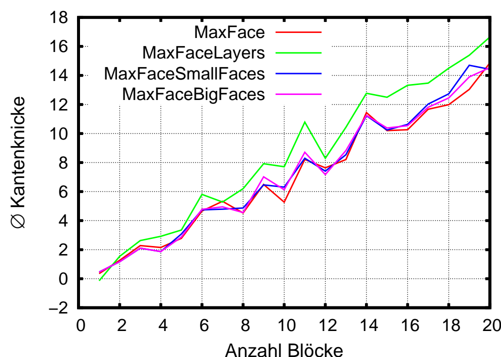


(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.

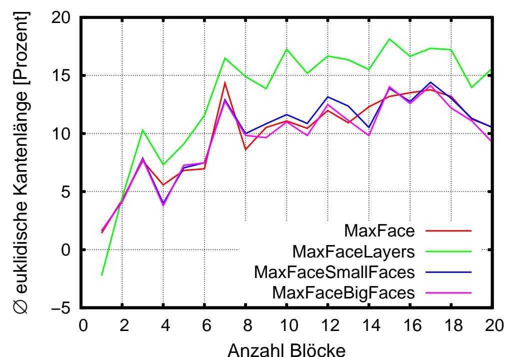


(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.

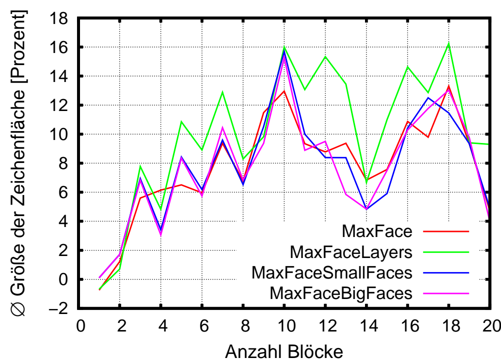
Abbildung 5.15: Vergleich der Varianten aus Kapitel 4 für Graphen aus der Rome-Library mit dem Algorithmus für maximale Außenfläche.



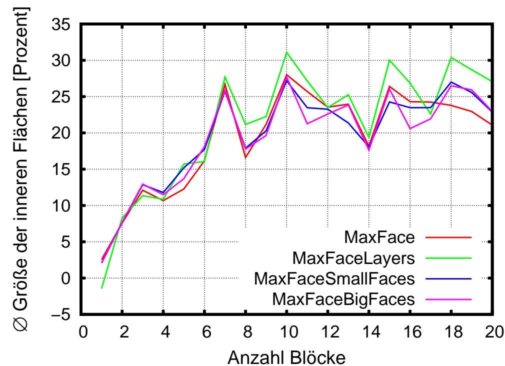
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.

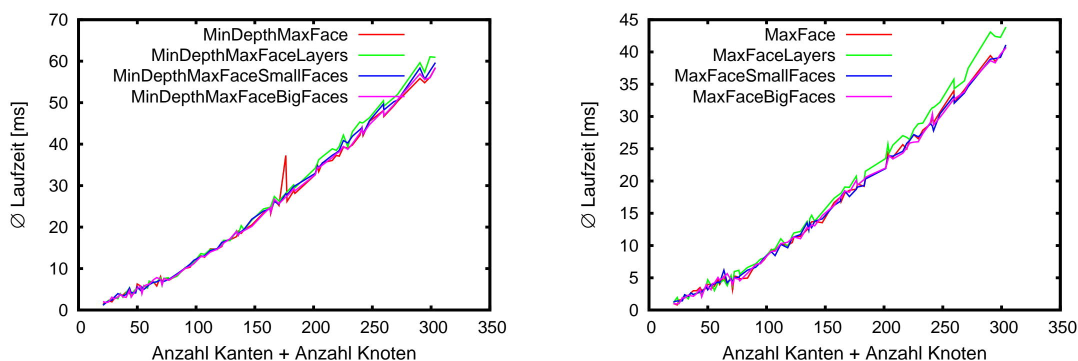


(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.



(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.

Abbildung 5.16: Vergleich der Varianten aus Kapitel 4 für Graphen aus der Block-Library mit dem Algorithmus für maximale Außenfläche.



(a) Varianten mit Algorithmus für Minimale Tiefe und Maximale Außenfläche

(b) Varianten mit Algorithmus für Maximale Außenfläche

Abbildung 5.17: Laufzeiten bei der Benutzung der verschiedenen Varianten.

Flächen für die Graphen aus der Block-Library erkennbar. MinDepthMaxFaceLayers schneidet bei der Größe der Zeichenfläche für Graphen mit 6 bis 9 Blöcken nicht am besten ab und kann bei der Größe der inneren Flächen erst ab Graphen mit 10 Blöcken deutlich besser abschneiden. Auch MaxFaceLayers kann sich bei der Größe der inneren Flächen erst ab Graphen mit 7 Blöcken von den anderen Algorithmen absetzen, mit einem Ausreißer bei Grahen mit 17 Blöcken, wo MaxFaceLayers schlechter abschneidet als MaxFace und MaxFaceSmallFaces.

In Kapitel 5.2 wurde bemerkt, dass keiner der dort getesteten Algorithmen für Graphen aus der Rome-Library mit mehr als 80 Knoten bei der Kantenknickanzahl besser abschneidet als Simple. Die Abbildungen 5.11(a) und 5.14(a) zeigen jedoch, dass die Algorithmen MinDepthMaxFaceLayers und MaxFaceLayers dies schaffen.

Die Abbildung 5.17 stellt die Laufzeit der verschiedenen Algorithmen und Varianten dar. Alle Varianten haben eine sehr ähnliche Laufzeit. Interessant ist dabei, dass die große äußere Schichten Varianten im Vergleich zu den anderen Varianten nur geringfügig mehr Rechenzeit in der Praxis benötigt. Denn für die Lösung des SSSP-Problems wurde ein Algorithmus mit Laufzeit $O(|V| \cdot |E|)$ verwendet (Bellman-Ford-Moore [BJG06]), obwohl das Problem in linearer Zeit gelöst werden kann [Tho99].

5.3.2 Beste Varianten im Vergleich mit dem Rest

Die Bezeichnungen der Algorithmen sind aus den vorigen Kapiteln übernommen worden. Die große äußere Schichten Variante, welche sowohl für den Algorithmus für maximale Außenfläche, als auch für den Algorithmus für minimale Tiefe und maximale Außenfläche, am besten abgeschnitten hat, wird nun mit den Algorithmen Simple, minimale Tiefe von Pizzonia und Tamassia und minimale erweiterte Tiefe von Gutwenger und Mutzel verglichen. Abbildung 5.19 stellt das Ergebnis für die Graphen aus der Rome-Library dar und Abbildung 5.20 das Ergebnis für die Graphen aus der Block-Library.

Wie zu erwarten, schneiden MaxFaceLayers und MinDepthMaxFaceLayers besser

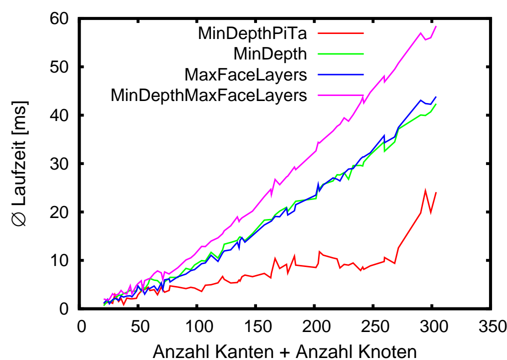
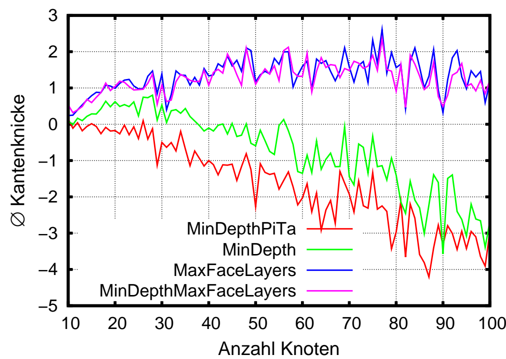


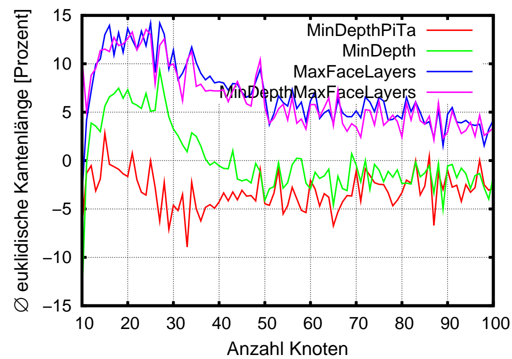
Abbildung 5.18: Messung der Laufzeit der Algorithmen.

als die anderen Algorithmen ab. Dies überrascht deshalb nicht, da MaxFace und MinDepthMaxFace besser abgeschnitten hatten als die anderen Algorithmen und MaxFaceLayers bzw. MinDepthMaxFaceLayers besser sind als MaxFace bzw. MinDepthMaxFace. Interessant ist aber zu sehen, wie deutlich der Unterschied gestiegen ist. Interessant ist auch, dass sich die Ergebnisse aus Kapitel 5.2 auch für die Layers Varianten übernehmen lassen. Für die Rome-Library sind MaxFaceLayers und MinDepthMaxFaceLayers gleich gut, bei den Graphen aus der Block-Library zeigt sich ein Vorteil von MinDepthMaxFaceLayers. Abbildung 5.18 zeigt die Laufzeiten der in diesem Kapitel getesteten Algorithmen. Das Ergebnis ist nahezu identisch zu den Laufzeiten aus Kapitel 5.2.

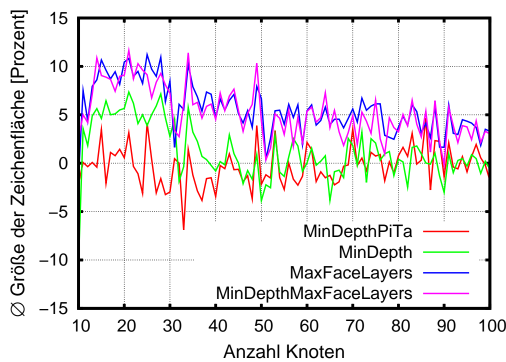
Insgesamt ergibt sich also die Empfehlung, für Graphen mit vielen Blöcken den Algorithmus für minimale Tiefe und maximale Außenfläche zu nutzen und dabei den Ansatz für große äußere Schichten zu verwenden. Für alle anderen Graphen eignet sich der Algorithmus für maximale Außenfläche mit dem Ansatz für große äußere Schichten aufgrund der geringeren Laufzeit am besten.



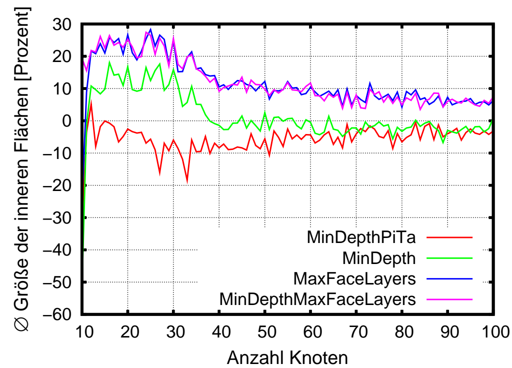
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.

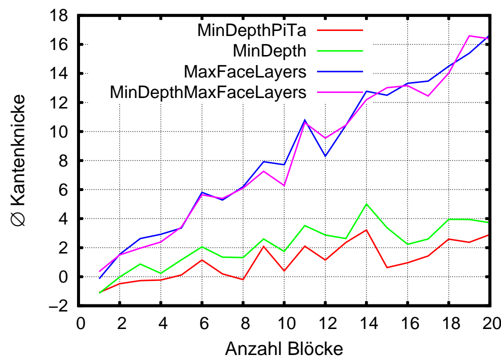


(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.

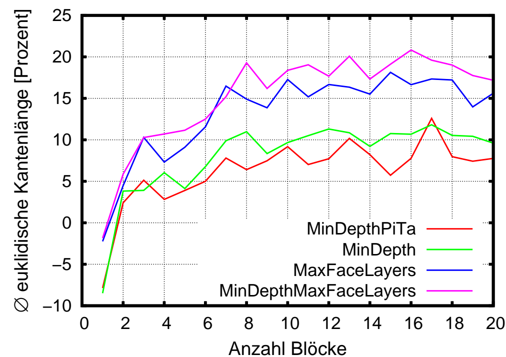


(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.

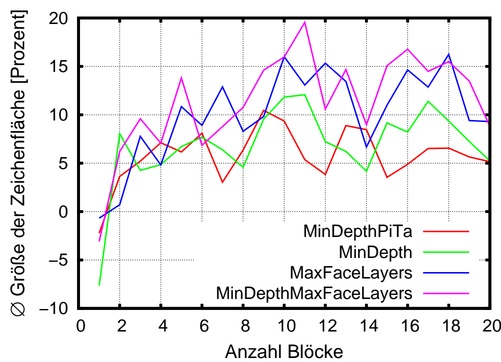
Abbildung 5.19: Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Rome-Library.



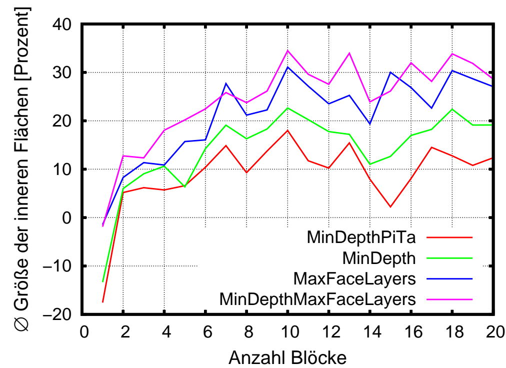
(a) Durchschnittliche Anzahl Kantenknicke: Bends(Simple) − Bends(Algorithmus).



(b) Durchschnittliche Summe der Kantenlängen: EdgeLength(Algorithmus) besser als EdgeLength(Simple), in Prozent.



(c) Durchschnittliche Größe der Zeichenfläche: Area(Algorithmus) besser als Area(Simple), in Prozent.



(d) Durchschnittliche Summe der Größe der inneren Flächen: InnerFaces(Algorithmus) besser als InnerFaces(Simple), in Prozent.

Abbildung 5.20: Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Block-Library.

6 Zusammenfassung und Ausblick

Nachdem in den vorigen Kapiteln alle Ergebnisse der Diplomarbeit im Detail beschrieben wurden, soll in diesem Kapitel eine kurze Zusammenfassung der wichtigsten Ergebnisse und Erkenntnisse gegeben werden. Außerdem werden noch Anregungen gegeben, welche weiterführenden Forschungen sich aus den Ergebnissen ergeben könnten.

6.1 Zusammenfassung

In der Diplomarbeit wurden Algorithmen behandelt, welche für einen planaren Graphen eine planare Einbettung berechnen. Der Topology-Shape-Metrics Ansatz (Kapitel 2.2) wurde benutzt, um Zeichnungen für Graphen zu erstellen, wobei die behandelten Algorithmen benutzt wurden. Es wurden zunächst die Algorithmen für *minimale Tiefe* von Pizzonia und Tamassia (Kapitel 3.1), *minimale Tiefe* von Gutwenger und Mutzel (Kapitel 3.3), *maximale Außenfläche* (Kapitel 3.2) sowie *minimale Tiefe und maximale Außenfläche* (Kapitel 3.4) beschrieben und die Algorithmenbeschreibungen aus den jeweiligen Papern teilweise um Details ergänzt.

Während der Diplomarbeit wurde bemerkt, dass der Algorithmus von Gutwenger und Mutzel für minimale Tiefe aus [GM04b] nicht die Tiefe minimiert, welche in dem Paper definiert wurde. Die Definition wurde angepasst und der Begriff der erweiterten Tiefe einer planaren Einbettung eingeführt. Jeder Algorithmus lässt sich leicht modifizieren, so dass er entweder die Tiefe oder die erweiterte Tiefe benutzt. So wurden die beiden minimale Tiefe Algorithmen auch jeweils in den Varianten für minimale Tiefe und minimale erweiterte Tiefe beschrieben. In Kapitel 5.1 wurde experimentell untersucht, welche der beiden Varianten in der Praxis bessere Ergebnisse liefert. Dabei wurde ein geringer Vorteil der Minimierung der erweiterten Tiefe festgestellt und im weiteren Verlauf immer die erweiterten Tiefe benutzt.

Die Zeichnungen, die entstehen, wenn die in Kapitel 3 vorgestellten Algorithmen im Topology-Shape-Metrics Ansatz benutzt werden, wurden experimentell miteinander verglichen (Kapitel 5.2). Dazu wurden zwei Graphensammlungen benutzt: zum einen reale Welt Graphen aus der Rome-Library und zum anderen Graphen, welche randomisiert mit steigender Blockanzahl erzeugt wurden (die so genannte Block-Library). Es wurde herausgefunden, dass für Graphen aus der Rome-Library die Algorithmen für *maximale Außenfläche* sowie *minimale Tiefe und maximale Außenfläche* die besten Ergebnisse liefern. Für die Graphen aus der Block-Library liefert der Algorithmus für *minimale Tiefe und maximale Außenfläche* etwas bessere Ergebnisse als der Algorithmen für *maximale Außenfläche*. Somit ist der Algorith-

mus für *minimale Tiefe und maximale Außenfläche* der Gesamtsieger. Der Algorithmus für *maximale Außenfläche* benötigt jedoch weniger Rechenzeit und liefert auch sehr gute Ergebnisse und kann somit auch empfohlen werden. Die Ergebnisse vom Algorithmus für *maximale Außenfläche* sind vor allem bei den Graphen aus der Rome-Library, also den reale Welt Graphen, sehr gut, da die Anzahl der Blöcke dort geringer und die topologische Verschachtelungstiefe geringer ist.

In Kapitel 4 wurden verschiedene Algorithmen vorgestellt, die sich mit den Algorithmen aus Kapitel 3 kombinieren lassen. Die Algorithmen aus Kapitel 3 bestimmen die Verschachtelung der Blöcke und das Äußere des Graphen bzw. der Blöcke des Graphen, die Algorithmen aus Kapitel 4 bestimmen das Innere. Es wurden drei neue Varianten für die Einbettung von Blöcken, welche nicht in die Außenfläche eingebettet werden können, vorgestellt. Die ersten beiden Varianten beschäftigen sich nur mit der Wahl der inneren Fläche, in die ein Block eingebettet wird, welcher nicht in die Außenfläche eingebettet werden kann. Die Einbettung der Blöcke wird aber nicht verändert. Die erste Variante (Kapitel 4.2) minimiert die maximale Größe einer inneren Fläche, wobei die Größe anhand der Knoten in den Blöcken, welche in diese Fläche eingebettet wird, gemessen wird. Die zweite Variante (Kapitel 4.3) versucht genau das Gegenteil. Anstatt viele kleine Flächen zu erzeugen, versucht sie, die Blöcke in möglichst wenige Flächen einzubetten, so dass wenige große Flächen entstehen. Es wurde für diese beiden Varianten jeweils eine Heuristik mit linearer Laufzeit beschrieben. Für die dritte Variante (Kapitel 4.4), welche neben der Einbettung von Blöcken in innere Flächen auch die Einbettung von inneren Kanten in Blöcken bestimmt, wurde ebenfalls eine Heuristik mit linearer Laufzeit beschrieben. In dieser Variante wird der Begriff der Schicht eingeführt und die lexikografisch größte Schichtenmenge berechnet. Anschaulich betrachtet bedeutet dies, dass die Größe der Flächen mit geringstem Abstand zur Außenfläche maximiert wird. Der Algorithmus benutzt die Knoten- und Kantenlängen, welche z.B. vom Algorithmus für *maximale Außenfläche* berechnet werden, und verändert nur den Schritt, in dem eine Einbettung berechnet wird. Dadurch lässt er sich ohne großen Aufwand mit den in Kapitel 3 vorgestellten Algorithmen kombinieren, berechnet aber nicht immer die optimale Lösung. Es wurde ein Beispiel gezeigt, in dem der Algorithmus unter Umständen nicht die optimale Lösung berechnen würde.

Die Varianten wurden in Kapitel 5.3 zuerst untereinander verglichen. Dazu wurden die beiden Sieger der vorigen Vergleiche, der Algorithmus für *maximale Außenfläche* und der Algorithmus für *minimale Tiefe und maximale Außenfläche*, in den drei neu vorgestellten Varianten sowie der alten Variante, die in der ersten Implementierung benutzt wurde (Kapitel 4.1), analysiert. Die Tests ergaben dabei einen klaren Sieg für die Variante der großen äußeren Schichten. Anschließend wurden die Tests aus Kapitel 5.2 noch einmal durchgeführt, wobei die Algorithmen für *maximale Außenfläche* sowie *minimale Tiefe und maximale Außenfläche* jeweils mit der Variante für große äußere Schichten kombiniert wurden. Die Ergebnisse sind vergleichbar mit denen des ersten Tests: die Algorithmen für *maximale Außenfläche* sowie *minimale Tiefe und maximale Außenfläche* sind wieder die Sieger, allerdings mit noch größerem Abstand als im ersten Test. Die Benutzung der große äußere Schichten Variante

verbessert die Ergebnisse allerdings für beide Algorithmen gleich stark, so dass die Algorithmen auch dieses Mal wieder bei den Graphen der Rome-Library ähnlich gute Ergebnisse liefern. Allerdings liefert auch bei diesem Test der Algorithmus für *minimale Tiefe und maximale Außenfläche* bei Graphen der Block-Library die besseren Ergebnisse. Die Laufzeit ist jedoch auch hier größer als die des Algorithmus für *maximale Außenfläche*, und da dieser auch gute Ergebnisse liefert, vor allem bei den Graphen der Rome-Library, kann auch er empfohlen werden. Die Laufzeit bei Benutzung der große äußere Schichten Variante steigt nur gering, somit können insgesamt die beiden Algorithmen für *maximale Außenfläche* sowie *minimale Tiefe und maximale Außenfläche* empfohlen werden, wobei sie jeweils mit der Variante für große äußere Schichten kombiniert werden sollten.

Das Gesamtfazit der Diplomarbeit ist, dass drei Algorithmen aus [GM04b], welche bisher nur theoretisch beschrieben waren, implementiert und analysiert wurden. Zwei dieser Algorithmen haben sich als eine Verbesserung im Vergleich mit einigen anderen Algorithmen herausgestellt. Zusätzlich wurde eine Erweiterung der Algorithmen entwickelt und implementiert, welche die Ergebnisse dieser beiden Algorithmen noch einmal verbessern.

6.2 Ausblick

Ein interessanter Vergleich wäre es, die Ergebnisse der beiden Sieger-Algorithmen mit einem Algorithmus wie z.B. [MW02] zu vergleichen, welcher die Kantenknickanzahl exakt minimiert. Dieser Test sollte in der Diplomarbeit schon gemacht werden, es lag aber leider keine Implementierung einer dieser Algorithmen im OGDF vor. Die Ergebnisse aus [MW02] lagen zwar vor, jedoch wurden die Graphen dort vor der Benutzung des Algorithmus planarisiert und auch durch Kanteneinfügungen zweizusammenhängend gemacht. Es konnte nicht sichergestellt werden, dass bei einer Planarisierung und einem Kanteneinfügen dieselben Graphen als Eingabe entstehen, welche für die Tests in [MW02] benutzt wurden. Und damit wäre nicht sicher gewesen, ob die Ergebnisse vergleichbar gewesen wären. Dieser Vergleich wäre aber vor allem deshalb interessant, da in den Experimenten in dieser Diplomarbeit festgestellt wurde, dass eine Zeichnung mit weniger Kantenknicken nicht immer besser ist als eine Zeichnung mit mehr Kantenknicken, wenn man auch Kriterien wie die Summe der Kantenlängen oder die Größe der benötigten Zeichenfläche betrachtet.

Weiterhin wäre eine Verbesserung der Laufzeit der Implementierung der Algorithmen möglich, welche die große äußere Schichten Variante benutzen. Für die Lösung des SSSP-Problems wird bei der Berechnung der Dicke einer R-Komponente, der Einbettung einer R-Komponente und für die Bestimmung der inneren Fläche, in die ein Block eingebettet wird, ein Algorithmus mit Worst-Case Rechenzeit $O(nm)$ verwendet (Bellman-Ford-Moore [BJG06]). Es existiert jedoch ein effizienterer Algorithmus mit linearer Rechenzeit [Tho99]. Da die Laufzeitmessungen jedoch ergeben haben, dass die Laufzeit bei der Benutzung der große äußere Schichten Variante nur geringfügig steigt, ist die erwartete Verbesserung der Laufzeit nicht sehr hoch.

Die beiden Algorithmen, welche in den Tests am besten abgeschnitten haben, könnten erweitert werden, um einen exakten Algorithmus für die Variante der großen äußeren Schichten zu berechnen. Dazu müsste die Berechnung der Knoten- und Kantenlängen angepasst werden (vergl. Kapitel 4.4.2). Die Ergebnisse ließen sich damit wahrscheinlich noch einmal ein wenig verbessern. Allerdings berechnet die vorgestellte und in den Tests benutzte Heuristik nur in bestimmten Fällen eventuell eine nicht optimale Lösung (vergl. Beispiel in Kapitel 4.4.2). Daher ist zu erwarten, dass sich die Ergebnisse für die Graphen aus der Rome-Library aufgrund ihrer Topologie nur geringfügig ändern. Es wäre aber interessant zu sehen, wie stark sich die Änderungen auf die Ergebnisse für die Graphen der Block-Library auswirken.

A Beiliegende DVD

Auf der DVD befinden sich die folgenden Dateien und Ordner:

- Datei `Diplomarbeit.pdf`: Dieses Dokument.
- Ordner `Source`: Beinhaltet ein Microsoft Visual Studio 2005 Projekt, welches das OGDF einbindet (dieses befindet sich im Unterverzeichnis `Src/OGDF`) und den C++ Quellcode beinhaltet, mit dem die Tests aus Kapitel 5 durchgeführt wurden (im Unterverzeichnis `Src/MinDepthMaxFace`). Es ist ebenfalls der Algorithmus enthalten, mit dem die Block-Library erzeugt wurde.
- Ordner `Rome-Pino`: Beinhaltet die Rome-Library, welche in Kapitel 5 benutzt wurde. In der zip-Datei `Raw` befindet sich die unveränderte Library und in den zip-Dateien `PlanarizedPrimalNodes` und `Planarized` die Graphen aus `Raw`, nachdem sie planarisiert wurden. In `PlanarizedPrimalNodes.zip` sind die Ordner `10nodes` bis `100nodes` enthalten und jeweils die Graphen, die vor der Planarisierung die jeweilige Knotenanzahl hatten. In `Planarized.zip` hingegen sind die Ordner `10nodes` bis `297nodes` und jeweils die Graphen in den Ordner, der zu der Knotenanzahl nach der Planarisierung passt.
- Ordner `Generated Graphs`: Beinhaltet die Graphen aus der Block-Library in der zip-Datei `BlockLibrary`.
- Ordner `Graph-Results`: Beinhaltet für jeden der in dieser Diplomarbeit vorgestellten Algorithmen und Varianten eine zip-Datei, z.B. `MinDepthMaxFaceLayers.zip`. Diese enthält die erzeugten Zeichnungen der Graphen der Rome-Library (Ordner `10` bis `100`) und der Graphen aus der Block-Library (Ordner `1Blocks` bis `20Blocks`) im GML-Format, welches im OGDF als Ausgabedateiformat für Graphen benutzt wird. Außerdem sind alle Ergebnisse der Tests in den Ordnern `Results`, `Results Blocks`, `Results Compare Depth Definitions` und `Results Compare Depth Definitions Blocks` enthalten. Die Diagramme in Kapitel 5 wurden mit dem Programm GNUplot erzeugt. Die Skripte dazu befinden sich in den Unterordnern `GNU-Plot Scripts` bzw. in der Datei `GNU-plot-All.plt` der Results-Ordner.
- Datei `gde-win.exe`: GoVisual Diagram Editor[1], mit dem sich Graphen im GML Dateiformat (s.o.) ansehen und bearbeiten lassen.

[1] `http://www.oreas.com/gde_en.php`, Stand: 16. August 2007

- Ordner `gnuplot4.0`: Das Programm GNUplot[2], s.o.

Auf der DVD befinden sich alle Programme in der Windows Version. Auf der jeweils angegebenen Webseite existieren aber auch Versionen für andere Betriebssysteme wie z.B. Linux.

[2] `http://www.gnuplot.info`, Stand: 16. August 2007

Literaturverzeichnis

[BE+98] Giuseppe Di Battista, Peter Eades, Roberto Tamassia und Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1. Aufl., 1998.

[BG+97] Giuseppe Di Battista, Ashim Garg, Giuseppe Liotta, Roberto Tamassia, Emanuele Tassinari und Francesco Vargiu. An experimental comparison of four graph drawing algorithms. *CGTA: Computational Geometry: Theory and Applications*, 7, 1997.

[BJG06] Jørgen Bang-Jensen und Gregory Gutin. *Digraphs: Theory, Algorithms, and Applications*, Kapitel 2.3.4, S. 55–58. Springer-Verlag, London, 4. Aufl., 2006.

[BLW86] Norman Biggs, Edward Keith Lloyd und Robin James Wilson. *Graph Theory: 1736-1936*. Clarendon Press, New York, NY, USA, 1986.

[BM76] John Adrian Bondy und Uppaluri Siva Ramachandra Murty. *Graph Theory with Applications*. Macmillan Press, New York, 1. Aufl., 1976.

[BT96a] Giuseppe Di Battista und Roberto Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15(4):302–318, 1996.

[BT96b] Giuseppe Di Battista und Roberto Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.

[Cha85] Gary Chartrand. *Introductory Graph Theory*. Dover Publications Inc, New York, 1985.

[Che76] Peter Pin-Shan Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CL04] Edward Corwin und Antonette Logar. Sorting in linear time – variations on the bucket sort. *Journal of Computing Sciences in Colleges*, 20(1):197–202, 2004.

[CN+85] Norishige Chiba, Takao Nishizeki, Shigenobu Abe und Takao Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985.

[Die05] Reinhard Diestel. *Graphentheorie*. Springer-Verlag, Heidelberg, 3. Aufl., 2005.

[DL98] Walter Didimo und Giuseppe Liotta. Computing orthogonal drawings in a variable embedding setting. In *ISAAC '98: Proceedings of the 9th International Symposium on Algorithms and Computation*, Band 1533 von *LNCS*, S. 79–88, London, UK, 1998. Springer-Verlag.

[EG+04] Markus Eiglsperger, Carsten Gutwenger, Michael Kaufmann, Joachim Kupke, Michael Jünger, Sebastian Leipert, Karsten Klein, Petra Mutzel und Martin Siebenhaller. Automatic layout of UML class diagrams in orthogonal style. *Information Visualization*, 3(3):189–208, 2004.

[FK96] Ulrich Fößmeier und Michael Kaufmann. Drawing high degree graphs with low bend numbers. In F.J. Brandenburg (Hrsg.), *Graph Drawing (Proc. GD '95)*, Band 1027 von *LNCS*, S. 254–266. Springer-Verlag, 1996.

[GDT] Graph Drawing Toolkit GDT. An object-oriented library for handling and drawing graphs. `http://www.dia.uniroma3.it/~gdt/`.

[GH+97] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*, Kapitel 5: Verhaltensmuster - Iterator, S. 300. Addison-Wesley (Deutschland), 2. Aufl., 1997.

[GM01] Carsten Gutwenger und Petra Mutzel. A linear time implementation of SPQR-trees. In Joe Marks (Hrsg.), *Graph Drawing, Colonial Williamsburg, 2000*, Band 1984 von *LNCS*, S. 77–90. Springer-Verlag, 2001.

[GM04a] Carsten Gutwenger und Petra Mutzel. An experimental study of crossing minimization heuristics. In Giuseppe Liotta (Hrsg.), *11th Symposium on Graph Drawing, 2003 Perugia*, Band 2912 von *LNCS*, S. 13–24. Springer-Verlag, 2004.

[GM04b] Carsten Gutwenger und Petra Mutzel. Graph embedding with minimum depth and maximum external face. In Giuseppe Liotta (Hrsg.), *11th Symposium on Graph Drawing, 2003 Perugia*, Band 2912 von *LNCS*, S. 259–272. Springer-Verlag, 2004.

[GMW05] Carsten Gutwenger, Petra Mutzel und René Weiskircher. Inserting an edge into a planar graph. *Algorithmica*, 41(4):289–308, 2005.

[GT01] Ashim Garg und Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing*, 31(2):601–625, 2001.

[Har69] Frank Harary. *Graph Theory*. Westview Press, 1969.

[Jun94] Dieter Jungnickel. *Graphen, Netzwerke und Algorithmen.* Spektrum Akademischer Verlag, Mannheim u.a., 3. Aufl., 1994.

[Knu98] Donald Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*, Band 3, Kapitel 5.2.4: Sorting by Merging, S. 158–168. Addison-Wesley Professional, 2. Aufl., 1998.

[Kro87] Lydia Kronsjö. *Algorithms: Their Complexity and Efficiency*, Kapitel Optimum Comparison Sorting, S. 232–234. John Wiley & Sons, Inc., New York, USA, 2. Aufl., 1987.

[LVB94] Giuseppe Liotta, Francesco Vargiu und Giuseppe Di Battista. Orthogonal drawings with the minimum number of bends. In *Proceedings of the 6th Canadian Conference on Computational Geometry*, S. 281–286. University of Saskatchewan, 1994.

[MW02] Petra Mutzel und René Weiskircher. Bend minimization in orthogonal drawings using integer programming. In *COCOON '02: Proceedings of the 8th Annual International Conference on Computing and Combinatorics*, S. 484–493, London, UK, 2002. Springer-Verlag.

[Piz01] Maurizio Pizzonia. *Engineering of Graph Drawing Algorithms for Applications.* Dissertation, Dipartimento di Informatica e Sistemistica, University degli Studi „La Sapienza“ di Roma, 2001.

[Piz05] Maurizio Pizzonia. Minimum depth graph embeddings and quality of the drawings: An experimental analysis. In *Graph Drawing*, Band 3843 von *LNCS*, S. 397–408, 2005.

[PT00] Maurizio Pizzonia und Roberto Tamassia. Minimum depth graph embedding. In *ESA 2000: Proceedings of the 8th Annual European Symposium on Algorithms*, Band 1879 von *LNCS*, S. 356–367, London, UK, 2000. Springer-Verlag.

[RJB04] James Rumbaugh, Ivar Jacobson und Grady Booch. *The Unified Modeling Language Reference Manual.* Addison Wesley, 2004.

[Tam87] Roberto Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[TBB88] Roberto Tamassia, Giuseppe Di Battista und Carlo Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):61–79, 1988.

[Tho99] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999.

[Tur04] Volker Turau. *Algorithmische Graphentheorie*. Oldenbourg-Verlag, München u.a., 2. Aufl., 2004.

[Wei02] René Weiskircher. *New Applications of SPQR-Trees in Graph Drawing*. Dissertation, Universität des Saarlandes, 2002.

Abbildungsverzeichnis

1.1 Beispiel für ein UML Klassendiagramm. 1
1.2 Einfluss der planaren Einbettung auf die Zeichnung für einen Beispielgraphen mit 23 Knoten. 2
1.3 Einfluss der planaren Einbettung auf die Zeichnung für einen Beispielgraphen mit 36 Knoten. 3

2.1 Schnittknoten und Blöcke. 9
2.2 Baum mit Wurzel v_0. 9
2.3 Graphen 10
2.4 Flächen der Zeichnung eines Graphen. 11
2.5 Topology-Shape-Metrics Ansatz. 12
2.6 Split-Operation 13
2.7 Beispiel dominieren von Split-Paaren 14
2.8 Struktur von zweizusammenhängenden Graphen und das Skelett der Wurzel des zugehörigen Proto-SPQR-Baums. 16
2.9 Beispiel SPQR-Baum. 18
2.10 Veranschaulichung einiger Begriffe im Zusammenhang mit SPQR-Bäumen. 19
2.11 Beispiel BC-Baum. 20
2.12 Ersetzen der Kante e durch die Multikanten e_1 und e_2. 21
2.13 Berechnung der Tiefe eines Graphen. 22
2.14 Vergleich Tiefe einer Einbettung mit erweiterter Tiefe einer Einbettung. 24

3.1 Beispiel für die Anzahl verschiedener Einbettungen. 26
3.2 Multikanten und Self Loops. 26
3.3 Ein Baum, sein Durchmesser- und Tiefenbaum. 30
3.4 Vergleich von zwei Einbettungen mit großer und minimaler Tiefe für einen reale Welt Graphen. 32
3.5 Vergleich von zwei Einbettungen mit großer und minimaler Tiefe für einen randomisiert erstellten Graphen. 33
3.6 Einfügen eines Dummy-Knotens mit zwei Dummy-Kanten. 34
3.7 Messungen für reale Welt Graphen. 35
3.8 Messungen für randomisiert erzeugte Graphen. 35
3.9 Expandieren von Fläche f_S zur Fläche f_Γ. 37
3.10 Ersetzen einer virtuellen Kante durch ihren Expansionsgraphen. . . . 39
3.11 Darstellung der Skelettgraphen mit fester Position der Referenzkante, p_ℓ und p_r. 39

3.12 Berechnen der Außenfläche bei gegebenem Adjazenzeintrag α. 40
3.13 Einbettung der Kanten eines inneren Knotens in einer R-Komponente. 43
3.14 Fixe Einbettung eines Blocks mit Platzierung von drei Graphen. . . . 43
3.15 Zusammenfügen verschiedener Block-Einbettungen zu Γ_G. 46
3.16 Zwei verschiedene Erweiterungen einer Einbettung. 48
3.17 Einfluss von Blöcken mit nur einer Kante auf die Tiefe. 52

4.1 Vergleich der Größe zweier Blöcke. 54
4.2 Güte der Heuristik für kleine innere Flächen. 56
4.3 Beispielgraph mit optimaler Lösungsgüte $n+1$ und Heuristiklösungsgüte $3n$. 56
4.4 Güte der Heuristik für wenige große innere Flächen. 58
4.5 Zwei Einbettungen mit unterschiedlichen Schichten. 61
4.6 Wahl der Einbettung einer R-Komponente. 62
4.7 Einbettung einer P-Komponente. 62
4.8 Wahl der Expansionsfläche für eine virtuelle Kante. 63
4.9 Dicke eines Skelettgraphen. 64
4.10 Beispiel für Algorithmus große äußere Schichten. 66
4.11 Einbettung der Kante e_i in einer P-Komponente. 67
4.12 Wahl der Expansionsfläche in einer R-Komponente. 68
4.13 Abstand von Flächen adjazent zu einem Schnittknoten zur Außenfläche. 70
4.14 Beispiel, in dem der Algorithmus unter Umständen versagt. 72

5.1 Eigenschaften der Testgraphensammlungen. 74
5.2 Beispiel für zwei Zeichnungen mit unterschiedlicher Größe der inneren Flächen. 75
5.3 Experimenteller Vergleich der verschiedenen Tiefendefinitionen für Graphen aus der Rome-Library. 76
5.4 Experimenteller Vergleich der verschiedenen Tiefendefinitionen für Graphen aus der Block-Library. 78
5.5 Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Rome-Library. 80
5.6 Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Block-Library. 82
5.7 Messung der Laufzeit der Algorithmen. 83
5.8 Planare Einbettung für G erstellt mit kleine Flächen Variante. 84
5.9 Planare Einbettung für G erstellt mit wenige große Flächen Variante. 85
5.10 Planare Einbettung für G erstellt mit große äußere Schichten Variante. 85
5.11 Vergleich der Varianten aus Kapitel 4 für Graphen aus der Rome-Library mit dem Algorithmus für minimale Tiefe und maximale Außenfläche. 87
5.12 Einbettung von Block B berechnet mit Algorithmus für maximale Außenfläche. 88

5.13 Einbettung von Block B berechnet mit Algorithmus für maximale Außenfläche mit großen äußeren Schichten. 89
5.14 Vergleich der Varianten aus Kapitel 4 für Graphen aus der Block-Library mit dem Algorithmus für minimale Tiefe und maximale Außenfläche. 90
5.15 Vergleich der Varianten aus Kapitel 4 für Graphen aus der Rome-Library mit dem Algorithmus für maximale Außenfläche. 91
5.16 Vergleich der Varianten aus Kapitel 4 für Graphen aus der Block-Library mit dem Algorithmus für maximale Außenfläche. 92
5.17 Laufzeiten bei der Benutzung der verschiedenen Varianten. 93
5.18 Messung der Laufzeit der Algorithmen. 94
5.19 Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Rome-Library. 95
5.20 Vergleich der Algorithmen aus Kapitel 3 für Graphen aus der Block-Library. 96