Praktikumsausarbeitung

Implementierung und Visualisierung des Planar-Separator-Algorithmus

Dirk Achenbach, Thomas Pajor, Jonida Saqe

{achen,pajor,saqe}@ira.uka.de

1. April 2007

Universität Karlsruhe (TH), Forschungsuniversität · gegründet 1825, Fakultät für Informatik, Institut für Theoretische Informatik, Algorithmik I, Prof. Dr. D. Wagner

Für Chantal-Jacqueline.

Im Rahmen des Praktikums „Algorithm Engineering“ im Wintersemester 2006/2007 an der Universität Karlsruhe haben wir den Algorithmus zum Planar-Separator-Theorem in Java implementiert und mit Hilfe von Java3D visualisiert. Das Planar-Separator-Theorem besagt, dass jeder planare Graph mit $n \geq 5$ Knoten durch Wegnahme einer Separatormenge, die weniger als $4 \cdot \sqrt{n}$ Knoten enthält, so in zwei nicht-zusammenhängende Komponenten zerteilt werden kann, dass jede höchstens $2n/3$ Knoten enthält. In dieser Ausarbeitung präsentieren wir unsere Implementierung des Algorithmus und möchten besonders auf die zum Teil nicht offensichtlichen Schwierigkeiten eingehen, die während der Implementierung aufgetreten sind. Abschließend stellen wir einige Ergebnisse von Experimenten bezüglich der Qualität des Algorithmus vor, die wir an drei Graphklassen durchgeführt haben.

1 Einleitung

Planare Graphen spielen eine wichtige Rolle in vielen Anwendungsgebieten wie etwa der Routenplanung oder zur Modellierung von Netzwerken. Eine besonders schöne Eigenschaft planarer Graphen ist, dass viele – im Allgemeinen schwierige – Algorithmen auf planaren Graphen effizient gelöst werden können. Als Zwischenschritt dieser Algorithmen wird oft das Aufteilen des Graphen in zwei ungefähr gleich große Hälften benötigt, auf die der Algorithmus rekursiv angewendet werden kann.

Es ist daher wichtig, diesen Aufteilungsschritt algorithmisch möglichst effizient lösen zu können. Lipton & Tarjan [LT77] haben gezeigt, dass dies für planare Graphen in $\mathcal{O}(n)$ Zeit möglich ist. Im Rahmen des Praktikums „Algorithm Engineering“, das auf der Vorlesung „Algorithmen auf planaren Graphen“ [Wag06] an der Universität Karlsruhe (TH) basiert, implementierten wir das Planar-Separator-Theorem von Lipton und Tarjan in Java. Der Schwerpunkt der Arbeit lag auf der Visualisierung des Algorithmus mit Hilfe von Java3D [jav] und dem Algovis3D-Framework [alg]. Zur Verwaltung der Graphdatenstruktur verwendeten wir die JungX-Bibliothek [junb], die die Jung-API [juna] um planare Graphen erweitert.

Experimente mit der Implementation des Algorithmus waren ebenfalls von zentraler Bedeutung für das Praktikum. Hierbei interessierte uns primär, wie sich der Algorithmus bezüglich einfacher Gütekriterien[1] auf verschiedenen Graphen diverser Graphklassen verhält.

Im nächsten Abschnitt gehen wir kurz auf das Theorem ein und wollen die dafür nötigen grundlegenden Begriffe definieren. Im Hauptteil der Ausarbeitung besprechen wir den Algorithmus im Detail. Im Laufe des Algorithmus werden einige Subalgorithmen, beispielsweise eine Triangulierung des Graphen, durchgeführt. Auf diese möchten wir ausführlicher eingehen, da die Implementierung dieser nicht ganz unproblematisch war. Abschließend werden wir noch einige Experimente vorstellen, die etwas über die Qualität des Algorithmus aussagen. Auf Laufzeitmessungen möchten wir aufgrund der Arbeitsumgebung Java verzichten.

2 Das Planar-Separator-Theorem

Ein Graph $G = (V, E)$ ist ein Tupel bestehend aus einer Menge von Knoten V, sowie einer Menge von Kanten $E \subseteq V \times V$. Eine Kante $e = \{u, v\} \in E$ verbindet also genau die Knoten u und v aus V. In diesem Fall heißen u und v *adjazent*, sowie e zu u bzw. v *inzident*. Wir wollen im Folgenden einen Graphen $G = (V, E)$ immer als ungerichtet[2], ohne Mehrfachkanten[3] und ohne Schleifen[4] betrachten.

Da der von uns implementierte Algorithmus nur auf planaren Graphen korrekt ablaufen kann,

[1] Wie beispielsweise die Separatorgröße oder die Ausgeglichenheit der Partition.
[2] Die Kanten haben keine vorgegebene Richtung.
[3] Zwischen je zwei Knoten gibt es höchstens eine Kante.
[4] Kein Knoten ist zu sich selbst adjazent.

wird der Begriff des *planaren Graphen* in dieser Ausarbeitung von zentraler Bedeutung sein. Deshalb möchten wir kurz auf die Definition eingehen.

Definition 1 (Planarer Graph)**.** *Ein Graph G heißt* planar*, wenn es eine Darstellung der Knoten und Kanten von G in der Ebene gibt, so dass sich keine zwei Kanten kreuzen. Die Knoten werden dabei auf Punkte in der Ebene, und die Kanten auf Jordan-Kurven zwischen den Punkten abgebildet.*

Eine konkrete Abbildung der Knoten und Kanten in die Ebene wollen wir *geometrische Einbettung* des Graphen nennen. Offensichtlich zerteilt eine solche Einbettung die Ebene in *Facetten* (Gebiete), die jeweils von Kanten bzw. Knoten begrenzt werden. Wir wollen die begrenzenden Knoten bzw. Kanten einer Facette f als die zu f inzidenten Knoten bzw. Kanten nennen. Mit dem Satz von Euler folgt, dass ein planarer Graph auf n Knoten höchstens $m = 3n-6$ Kanten haben kann; siehe dazu auch [Die06], Seite 99–100. In einem solchen maximal planaren Graphen sind alle Facetten durch genau drei Kanten begrenzt, und wir wollen diesen Graphen als *trianguliert* bezeichnen. Jeder nicht notwendigerweise maximal planare Graph $G = (V, E)$ kann durch geeignetes Einfügen von Kanten zu einem maximal planaren Graphen $G' = (V, E')$ *trianguliert* werden. Dies wird später als wichtiger Zwischenschritt in unserem Algorithmus eine Rolle spielen.

Lipton und Tarjan [LT77] haben 1977 gezeigt, dass für jeden planaren Graphen mit mehr als fünf Knoten stets eine Menge S von Knoten so gefunden werden kann, dass der Graph durch Wegnahme von S und allen zu S inzidenten Kanten in zwei Teile zerfällt, die „gut“ balanciert sind; siehe auch Abbildung 1 zur Verdeutlichung. Was das im Genauen bedeutet, liefert das *Planar-Separator-Theorem* im Folgenden.

Theorem 1 (Lipton & Tarjan, 1977)**.** *Sei $G = (V, E)$, $n \geq 5$ ein zusammenhängender und planarer Graph. Der Graph kann so in $V_1, V_2, S \subseteq V$ partitioniert werden, dass gilt*

- $|V_1|, |V_2| \leq \frac{2}{3}n$
- *S ist ein Separator, der V_1 und V_2 trennt*
- $|S| \leq 4\sqrt{n}$*.*

V_1, V_2, S können in $\mathcal{O}(n)$ Zeit gefunden werden.

Der Beweis des Theorems liefert bereits einen Algorithmus zur Bestimmung der Mengen S sowie V_1 und V_2, der linearen Zeitaufwand in der Anzahl der Knoten hat. Lipton und Tarjan garantieren sogar eine schärfere Schranke von $|S| \leq 2\sqrt{2n}$ für die Separatorgröße; unsere Implementierung orientiert sich jedoch im Wesentlichen an [Wag06], und wir können nur die o.g. Schranke garantieren.

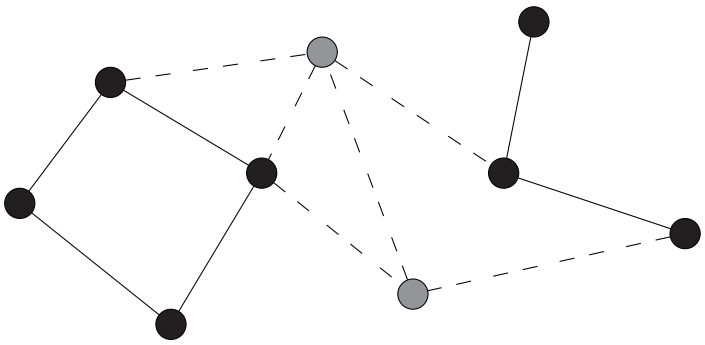


Abbildung 1: Planarer Graph, der durch Löschen der hellen Knoten und derer inzidenter, gestrichelter Kanten in zwei Teile separiert wird.

3 Der Algorithmus

In diesem Teil möchten wir detaillierter auf den Algorithmus eingehen. Wir werden nicht alle Schritte (wie zum Beispiel das Aufbauen eines Breitensuchbaums auf dem Graphen) ausführlich beschreiben, da dies den Rahmen dieser Ausarbeitung sprengen würde. Es gibt jedoch – auf den ersten Blick triviale – Stellen im Laufe des Algorithmus, die sich bei der konkreten Implementierung als problematisch erwiesen haben; daher werden wir auf diese besonders eingehen.

Der Algorithmus besteht aus drei Phasen, die wir in den nächsten Abschnitten jeweils für sich vorstellen werden. Die Phasen bauen dahingehend aufeinander auf, dass erzeugte Datenstrukturen und gesammelte Informationen in Folgephasen übernommen werden. Gelingt es einer Phase, einen geeigneten Separator zu finden, bricht der Algorithmus ab; sonst wird mit der Nachfolgephase fortgefahren. Phase III liefert immer einen korrekten Separator. Zunächst sei hier eine grobe Übersicht über den Ablauf des Gesamtalgorithmus und seiner Phasen gegeben.

- **Phase I.** Baue einen Breitensuchbaum auf dem Graphen auf. Finde den „mittleren Level“ μ in dem Baum und prüfe, ob dieser bereits einen geeigneten Separator liefert, der weniger als $4\sqrt{n}$ Knoten enthält.
- **Phase II.** Ist das nicht der Fall, suche ausgehend von dem mittleren Level den ersten Level oberhalb bzw. unterhalb von μ, der klein genug ist, und nenne diese m bzw. M. Prüfe, ob die Level m und M einen geeigneten Separator bilden.
- **Phase III.** Verschmelze alle Knoten unterhalb von Level m, und lösche alle Knoten oberhalb von Level M. Trianguliere den Graphen und führe eine Fundamentalkreissuche durch. Das heißt, vergrößere bzw. verkleinere solange einen durch eine Nichtbaumkante induzierten Kreis, bis ein Kreis gefunden wurde, der einen Separator liefert, der die Knoten im Innern von den Knoten außerhalb des Kreises trennt.

Auf die genaue Definition der Begriffe und Algorithmen werden wir in den folgenden Abschnitten ausführlich eingehen.

Bevor wir nun mit Phase I fortfahren, möchten wir noch kurz auf einen Vorverarbeitungsschritt eingehen. In Abschnitt 2 haben wir erläutert, dass die Eingabe für den Algorithmus eine geometrische Einbettung des planaren Graphen ist. Da wir jedoch im Wesentlichen nur die Kantenfolge an den Knoten als Information benötigen, ist es bequemer mit einer *kombinatorischen Einbettung* zu arbeiten.

Definition 2. *Sei $G = (V, E)$ ein planarer Graph. Eine* kombinatorische Einbettung *definiert für jeden Knoten $v \in V$ eine zirkuläre Ordnung der zu v inzidenten Kanten. Ist $e \in E$ mit $v \in e$, so ist $\Theta_v(e)$ die an v gegen den Uhrzeigersinn nächste Kante, während $\Theta_v^{-1}(e)$ die im Uhrzeigersinn nächste Kante an v ist.*

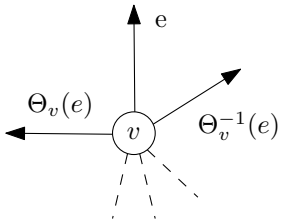


Abbildung 2: Kantenfolge an Knoten v bezüglich Kante e.

Zur Verdeutlichung dieser Definition siehe auch Abbildung 2. Algorithmus 1 erzeugt aus einer gegebenen geometrischen Einbettung eines Graphen eine kombinatorische Einbettung und ist Teil der Vorverarbeitung für den Hauptalgorithmus.

Algorithmus 1 : EMBEDDER

Eingabe : Planarer Graph $G = (V, E)$ in geometrischer Einbettung
Ausgabe : Planarer Graph $G' = (V', E')$ in kombinatorischer Einbettung

```
1 for v ∈ V do
2     v' ← v
3     Füge v' in G' ein
4 for e = {v1, v2} ∈ E do
5     e' ← {v1, v2}
6     vor1 ← FindeVorgänger(v1, e')
7     vor2 ← FindeVorgänger(v2, e')
8     Füge e' unter Berücksichtigung von vor1 und vor2 in G' ein
```

Die kombinatorische Einbettung von G können wir nun benutzen, um mit Phase I unseres eigentlichen Algorithmus zu beginnen.

3.1 Phase I

Phase I des Algorithmus schafft die Grundlage für alle nachfolgenden Operationen. Im ersten Schritt erzeugen wir – wie bereits in der Übersicht zu sehen ist – einen Breitensuchbaum T der Höhe h auf G. Dieser spannt den Graphen auf. Die Wahl der Wurzel für den Breitensuchbaum ist zunächst nicht festgelegt, kann aber zu Variationen in der Laufzeit des Algorithmus führen. Wir betrachten in unserer Implementierung einen beliebigen Knoten des Graphen als Startknoten für die Breitensuche.

Im Zusammenhang mit einem Breitensuchbaum (BFS-Baum), werden wir den den Begriff des *Level*s eines Knoten des Öfteren gebrauchen:

Definition 3 (Level eines Knoten)**.** *In einem Breitensuchbaum T wird die Länge des Pfades von einem Knoten v zur Wurzel des Baums T als das* Level des Knoten v *bezeichnet.*

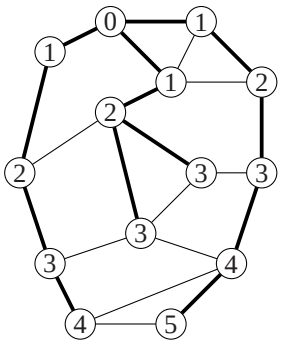


Abbildung 3: Ein aufspannender Breitensuchbaum auf einem Graphen inklusive Markierung der Levels.

Nun fassen wir alle Knoten eines Levels i in einer Menge S_i zusammen. S_0 enthält damit ausschließlich den Wurzelknoten des Breitensuchbaums, S_1 enthält alle zum Wurzelknoten adjazenten Knoten, usw.

Ein erster Separatorkandidat wird dadurch erzeugt, dass wir ein $\mu \in \{0, \ldots, h\}$ derart finden, dass gilt:

$$\sum_{i=0}^{\mu-1} |S_i| \leq \frac{n}{2} \quad \text{und}$$
$$\sum_{i=0}^{\mu} |S_i| > \frac{n}{2}$$

Wir suchen μ also derart, dass der μ-Level „knapp über der Hälfte der Knoten liegt“. Algorithmus 2 illustriert unser Vorgehen zum Finden des μ-Levels. Im Wesentlichen iterieren wir von der Wurzel levelweise durch den Graphen und addieren die Kardinalitäten der Knoten eines Levels auf, bis wir den Level μ gefunden haben.

Algorithmus 2 : FINDEMU

Ausgabe : „Mittlerer“ Level μ

```
1 für i ← 0 ... h tue
2     wenn Σ_{j=0}^{i-1} |S_j| ≤ n/2 und Σ_{j=0}^{i} |S_j| > n/2 dann
3         μ ← i
4         bruch
5 return μ
```

Gilt nun $|S_\mu| \leq 4\sqrt{n}$ und $\mu < h$, ist mit dem μ-Level bereits ein gültiger Separator gefunden:

$$S := S_\mu$$
$$V_1 := \bigcup_{i=0}^{\mu-1} S_i$$
$$V_2 := \bigcup_{i=\mu+1}^{h} S_i$$

Erfüllt μ die vom Theorem geforderten Bedingungen nicht, so müssen wir in Phase II übergehen.

3.2 Phase II

In Phase II finden wir zusätzlich zum μ-Level noch einen m- und M-Level und prüfen, ob letztere einen korrekten Separator induzieren.

Wir finden m so, dass S_m der unterste Level oberhalb von S_μ ist, der weniger als $\sqrt{n}$ Knoten enthält – oder genausoviele. Es liegt nahe, dass wir von μ aus in Richtung Wurzel iterieren und abbrechen, sobald der aktuell betrachtete Level klein genug ist, um m zu finden. M ist analog zu m als der oberste Level unterhalb von S_μ definiert, für den $|S_M| \leq \sqrt{n}$. Wir finden M ebenso wie m. Man muss hier beachten, dass es durchaus möglich ist, dass $M = h + 1$ und damit $S_M = \emptyset$.

Mithilfe von μ, m und M können wir uns nun drei Mengen A_1, A_2 und A_3 konstruieren:

$$A_1 := \bigcup_{i=0}^{m-1} S_i$$
$$A_2 := \bigcup_{i=m+1}^{M-1} S_i$$
$$A_3 := \bigcup_{i=M+1}^{h} S_i$$

Der ursprüngliche Graph besteht nun also aus fünf „Schichten“, die entlang des Breitensuchbaums verlaufen: A_1, S_m, A_2, S_M, A_3.

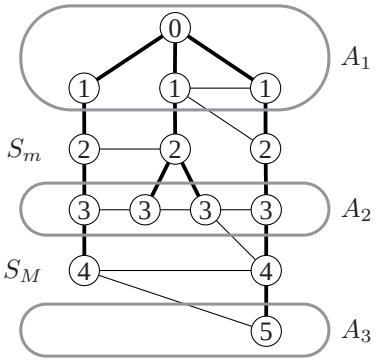


Abbildung 4: A_1, S_m, A_2, S_M und A_3.

Ist $|A_2| \leq \frac{2}{3}n$, so haben wir mit $S_m \cup S_M$ einen Separator gefunden:

$$S := S_m \cup S_M$$
$$V_1 \in \{A_1, A_2, A_3\}, \text{kardinalitätsmaximal}$$
$$V_2 := V \setminus (V_1 \cup S)$$

Ist aber $|A_2| > \frac{2}{3}n$, so gehen wir in Phase III über.

3.3 Phase III

Phase III ist von allen drei Phasen die aufwändigste, sie führt jedoch in jedem Fall zu einem Ergebnis, das den Bedingungen von Theorem 1 entspricht. Wir betrachten nicht länger die Level des BFS-Baums als Separatorkandidaten, sondern suchen Kreise im Graphen, die dann zu einem Separator führen werden.

Definition 4 (Fundamentalkreis)**.** *Sei $G = (V, E)$ ein Graph und $T \subset E$ ein spannender Baum auf G. Zu einer Nichtbaumkante $e \in E$, $e \notin T$ heißt der in T durch e induzierte Kreis $C_e \subset V$* Fundamentalkreis zur Kante e.

Die Knoten aus C_e bilden einen Separator in G, der die Knoten aus dem Inneren des Kreises von den Knoten außerhalb des Kreises trennt. Die Idee in Phase III ist, beginnend mit einem beliebigem Fundamentalkreis, durch sukzessives „Vergrößern“ und „Verkleinern“ des Kreises, zu einem Fundamentalkreis zu gelangen, der den Graphen gemäß den Bedingungen von Theorem 1 separiert. Wir wollen das Verfahren *Fundamentalkreissuche*, und den Separator *Fundamentalkreisseparator* nennen.

Da wir den Graphen aber nur in seiner kombinatorischen Einbettung betrachten und damit keine Lageinformationen zur Verfügung haben, ist die Unterscheidung zwischen Kreisinnerem und -äußerem nicht mehr offensichtlich.
Tatsächlich ist es aber irrelevant, was Inneres und Äußeres ist, wenn der Wurzelknoten des BFS-Baums selbst Teil des Kreises C_e ist. Stellt man sich den Graphen auf einer Kugel eingebettet vor[5], sieht man leicht ein, dass sich Kreisinneres und -äußeres nicht voneinander unterscheiden lassen. Es ist für unseren Algorithmus also unerheblich, ob die Knotenmenge, die wir als „Inneres“ abzählen tatsächlich geometrisch innen liegt – zumindest, solange der BFS-Wurzelknoten Teil von C_e ist und wir bei zwei Abzählungen immer dieselbe Seite als „innen“ abzählen.
Liegt der BFS-Wurzelknoten nicht auf dem Kreis, so ist der Pfad vom höchsten (bzgl. des BFS-Baumes) Kreisknoten zur Wurzel des BFS-Baums eindeutig und liegt außerhalb des Kreises. In diesem Fall ist also eindeutig bestimmt, was Kreisinneres bzw. -äußeres ist.

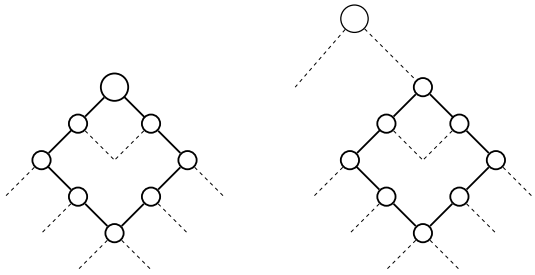


Abbildung 5: Kreisinneres vs. -äußeres mit BFS-Wurzelknoten $\in C_e$ und $\notin C_e$.

Die Fundamentalkreissuche direkt auf G auszuführen, kann unter Umständen einen Separator finden, der zu groß ist, da – wegen der Höhe h des BFS-Baums – eine obere Schranke für die Anzahl Knoten in einem Kreis C sicherlich $|C| \leq 2h + 1$ ist. Wegen $h \in \mathcal{O}(n)$ wäre ein solcher Separator im Allgemeinen nicht hinreichend klein, um weniger als $4\sqrt{n}$ Knoten zu beinhalten. Um dem entgegenzukommen, löschen wir alle Knoten $\bigcup_{i=M}^{h} S_i$ aus dem Graphen und verschmelzen die Knoten aus $\bigcup_{i=0}^{m} S_i$ zu einem neuen Knoten s durch sukzessives Zusammenziehen von Kanten. Der Knoten s bildet dann die neue Wurzel unseres BFS-Baums

[5] Eine Einbettung auf einer Kugeloberfläche ist äquivalent zu einer Einbettung auf einer Fläche.

auf dem kontrahierten Graphen $G' = (V', E')$.

Damit der Vergößerungs- bzw. Verkleinerungsschritt immer durchgeführt werden kann, muss der kontrahierte Graph G' noch trianguliert werden. Das heißt, es werden so viele Kanten in G' eingefügt, bis G' ein maximal planarer Graph ist und somit alle Facetten Dreiecke bilden. Das *Fundamentalkreislemma* garantiert uns nun die Existenz eines geeigneten Kreises.

Lemma 1 (Fundamentalkreislemma)**.** *Sei $G = (V, E)$ ein maximal planarer Graph mit $|V| \geq 5$, und T ein aufspannender Baum von G mit Höhe h. Dann existiert ein Fundamentalkreis C, der folgende Eigenschaften hat:*

i) C *trennt* $\mathrm{Inneres}(C)$ *von* $\ddot{\mathrm{A}}\mathrm{u\ss eres}(C)$

ii) $\mathrm{Inneres}(C), \ddot{\mathrm{A}}\mathrm{u\ss eres}(C) \leq \frac{2}{3}n$

iii) $|C| \leq 2h + 1$

Ein solcher Kreis kann in $\mathcal{O}(n)$ gefunden werden.

Beweis. Siehe [Wag06]. □

Wegen der Kontraktion und des Löschens der Level gilt, dass der gefundene Kreis in dem ursprünglichen Graphen nicht mehr als $2\sqrt{n}$ Knoten enthält.

Wir werden im weiteren Verlauf dieses Abschnitts detailliert auf das Verschmelzen der Knoten, die Triangulierung sowie die Fundamentalkreissuche eingehen. Da am Schluss der ursprüngliche Graph wiederhergestellt werden muss, ergaben sich – bedingt durch das Framework – einige Schwierigkeiten bei der Implementierung der Verschmelzung, die wir jedoch mit ein paar Tricks lösen konnten.

3.3.1 Verschmelzen und Löschen der Level

Die Knoten in $A_1 \cup S_m$ werden in diesem Schritt verschmolzen und $S_M \cup A_3$[6] gelöscht. In [Wag06] ist vorgesehen, diesen Schritt nach der Triangulierung (s.u.) durchzuführen. Wir ziehen ihn allerdings vor, weil wir die Triangulierung sonst nach dem Löschen von $S_M \cup A_3$ wiederherstellen müssten. Diese Reihenfolgenvertauschung hat keine weiteren Seiteneffekte.

Wir implementierten den Verschmelzungsschritt so, dass wir die Knoten nach und nach derart verschmelzen, dass ein Knoten in seinen Vorgänger bezüglich des BFS-Baums geschmolzen wird. Hierdurch ist das Problem der Mengenverschmelzung auf mehrere Einzelverschmelzungen reduziert.

Der Algorithmus sieht vor, dass der Ursprungsgraph nach seiner Durchführung wieder zur Verfügung steht. Da es das Framework nicht unterstützt, dass man den Graphen während der Algorithmenausführung austauscht, können wir nicht auf einem temporären Graphen arbeiten.

[6] Selbstverständlich inklusive der inzidenten Kanten.

Wir sind stattdessen gezwungen, gelöschte Knoten und Kanten später wieder in den Graphen einzufügen und eingefügte Kanten zu löschen[7]. Beim Einfügen von vormals gelöschten Kanten muss allerdings bedacht werden, dass die zu dieser Kante inzidenten Knoten zum Einfügezeitpunkt auch schon im Graphen vorhanden sein müssen. Weiterhin gibt es Kanten, die sich aufgrund der Verschmelzung nur „vorübergehend“ im Graphen befinden, also nicht gelöscht werden dürfen, bevor sie eingefügt werden.

Wir lösen dieses Problem dadurch, dass wir jeweils einen Keller für gelöschte Kanten, sowie für eingefügte Kanten und Knoten pflegen. Jede Graphmanipulation wird auf einem der Keller festgehalten. Zum Algorithmenende hin werden die durchgeführten Operationen in exakt umgekehrter Reihenfolge durchgeführt.

3.3.2 Triangulierung

Ist $G = (V, E)$ ein planarer Graph, so wollen wir, wie in Abschnitt 2 erwähnt, G trianguliert nennen, falls alle Facetten in G ein Dreieck bilden. Die Triangulierung hat also die Aufgabe, geeignete Kanten so einzufügen, dass neue Kanten

i) keine Doppelkanten induzieren;

ii) die Planaritätseigenschaft des Graphen nicht zerstören. Das heißt, es darf keine Kreuzung mit einer anderen Kante entstehen.

Die grundlegende Idee des Triangulierungsalgorithmus ist die folgende. Wir betrachten für jeden Knoten v alle inzidenten Kanten e im Uhrzeigersinn. Es werden nun, falls möglich, Kanten so eingefügt, dass die Facette rechts von e nach Einfügen der Kanten ein Dreieck bildet. Da der Algorithmus über alle Knoten und dort wiederum über alle inzidenten Kanten iteriert, wird jede Kante höchstens zweimal betrachtet. Geht man davon aus, dass das Einfügen einer Kante in konstanter Zeit möglich ist, so hat die Triangulierung eine Laufzeit von $\mathcal{O}(2m+n) = \mathcal{O}(n)$.

Das Bestimmen, wie eine Kante eingefügt wird, geschieht durch eine Fallunterscheidung der Konfigurationen, die die Kanten $e, \Theta^{-1}(e), \Theta^*(e)$ und $\Theta^*(\Theta^*(e))$ einnehmen können. Die Details zur Fallunterscheidung sind in [Paj07] näher beschrieben.

3.3.3 Fundamentalkreissuche

Die Fundamentalkreissuche bildet das eigentliche Kernstück der dritten Phase. Algorithmus 3 illustriert den prinzipiellen Aufbau, wobei wir auf den Verkleinerungs- bzw. Vergrößerungsschritt später eingehen möchten. Die Wahl der ersten Nichtbaumkante – und somit des ersten Kreises – ist nicht festgelegt, kann aber möglicherweise Einfluss auf die Qualität des Ergebnisses und die Geschwindigkeit des Algorithmus haben. Unsere Implementierung wählt eine

[7] Dies betrifft auch die Triangulierungskanten.

beliebige Nichtbaumkante. Das Fundamentalkreislemma 1 garantiert, dass die Schleife in Algorithmus 3 in jedem Fall nach endlicher Zeit terminiert.

Algorithmus 3 : FUNDAMENTALKREISSUCHE

Eingabe : Triangulierter, planerer Graph $G = (V, E)$ mit BFS-Baum T

```
1  e ← beliebige Nichtbaumkante in G
2  solange wahr tue
3      C ← Fundamentalkreis induziert durch e
4      wenn Inneres(C) ≥ Äußeres(C) und Inneres(C) ≤ 2/3n dann
5          bruch
6      sonst wenn Inneres(C) ≥ Äußeres(C) und Inneres(C) > 2/3n dann
7          Verkleinere C
8      sonst wenn Äußeres(C) ≥ Inneres(C) und Äußeres(C) ≤ 2/3n dann
9          bruch
10     sonst
11         Vergrößere C
12     Setze Separator S sowie V1 und V2 zusammen
```

Mit $\mathrm{Inneres}(C)$ bzw. $\mathrm{Äußeres}(C)$ bezeichnen wir die Knoten im Inneren bzw. Äußeren des Kreises. Es gilt $|V| = |\mathrm{Inneres}(C)| + |\mathrm{Äußeres}(C)| + |C|$ für jeden Fundamentalkreis C. Es reicht also, für jeden Kreis nur die Knoten im Inneren zu berechnen, und mithilfe dieser die Anzahl der Knoten im Äußeren abzuleiten. Um die Knoten im Inneren effizient zählen zu können, berechnen wir beim Aufbau des Breitensuchbaums in $\mathcal{O}(n)$ für jeden Knoten die Größe seines Unterbaums. Diese Information benutzen wir zur Bestimmung der Größe des Inneren des Kreises, indem wir alle Kreisknoten ablaufen und die Größen der Unterbäume dieser Knoten aufaddieren. Siehe dazu auch Abbildung 6.

Sind nun Inneres und Äußeres so balanciert, dass Fall 1 (Zeile 3) oder Fall 2 (Zeile 7) eintreten, so können wir die Fundamentalkreissuche abbrechen, und den Separator aus dem gefundenen Kreis zusammensetzen; siehe dazu den nächsten Abschnitt. Andernfalls müssen wir den Kreis solange verkleinern bzw. vergrößern, bis Inneres und Äußeres im geforderten Gleichgewicht stehen.

Wir betrachten hier o.B.d.A. einen *Verkleinerungsschritt*. Der Vergrößerungsschritt ist symmetrisch und bedarf in der Implementierung lediglich einer Fallunterscheidung. Wir möchten es vermeiden, den Kreis wahllos zu verkleinern, da dies dazu führen könnte, dass der Algorithmus zwischen einem Verkleinerungs- und Vergößerungsschrit hin- und herspringt, und somit das Terminieren der Fundamentalkreissuche nicht mehr sichergestellt wäre. Um den Kreis kontrolliert zu verkleinern, bedarf es der Triangulierung, da sie Voraussetzung für das folgende Lemma ist:

Lemma 2. *Sei $G = (V, E)$ ein planarer, triangulierter Graph und T ein BFS-Baum auf G.*

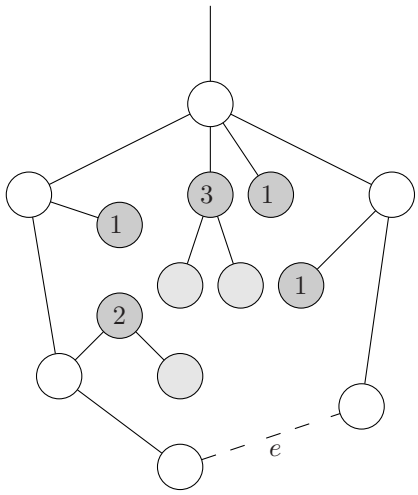


Abbildung 6: Fundamentalkreis C_e. Wir zählen für jeden Knoten $v \in C_e$ die größe seiner Unterbäume im Inneren des Kreises auf. Die Größen der Unterbäume sind mit den Zahlen der dunkelgrauen Knoten angedeutet. Da der Baum aufspannend ist, erhalten wir so die Anzahl Knoten im Inneren von C_e.

Dann gilt für jede Kante $\{u, v\} \in E$

$$|\operatorname{Level}(u) - \operatorname{Level}(v)| \leq 1$$

Damit folgt: Ist $e := \{x, y\}$ eine Nichtbaumkante des Kreises C_e mit $\operatorname{Inneres}(C_e) > 0$, so gibt es immer einen Knoten t im Inneren von C_e, der sowohl zu x als auch zu y inzident ist – siehe Abbildung 7.

Ist $t \in C_e$, wobei o.B.d.A. $\{t, x\}$ die Baumkante ist, so wählen wir $\{t, y\}$ als neue Nichtbaumkante. Die Facette, die von den Knoten t, x und y begrenzt wird, liegt nun im Äußeren des Kreises C_{ty}. Somit ist der Kreis um genau eine Facette verkleinert worden. Ist andererseits $t \notin C_e$, dann berechnen wir die Kreise C_{tx} und C_{ty}, und wählen als neuen zu betrachtenden Fundamentalkreis denjenigen mit größerem Inneren. Sei o.B.d.A. C_{tx} der größere der beiden Kreise. Wegen Lemma 2 gilt

$$\operatorname{Inneres}(C_{tx}) = \operatorname{Inneres}(C_e) - 1$$

was ebenfalls dazu führt, dass in einem Verkleinerungsschritt das Äußere des Kreises nicht wesentlich zu groß wird. Aus Effizienzgründen reichen wir die Information, auf welcher Seite von e das Innere des Kreises liegt, an den verkleinerten Fundamentalkreis weiter. Somit ersparen wir uns die für den neuen Kreis anfallende Berechnung zur Bestimmung des Inneren, und können sofort mit dem Akkumulieren der Unterbaumgrößen fortfahren.

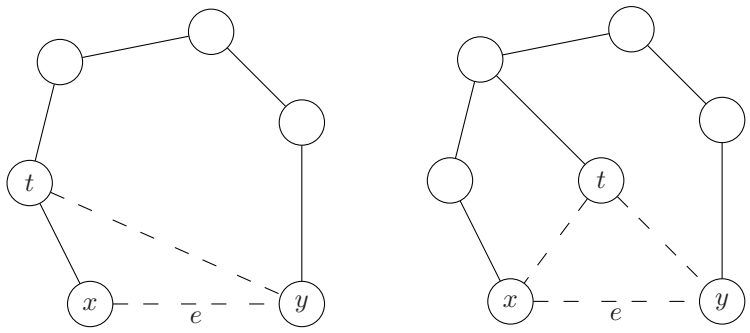


Abbildung 7: Wegen der Triangulierung existiert immer ein Knoten t im Inneren von C_e, der zu x und y adjazent ist. Es ist möglich, dass t selbst auf dem Kreis liegt (links), dann ist mindestens eine der Kanten $\{t, x\}$ oder $\{t, y\}$ eine Baumkante. Im anderen Fall (rechts) sind beide Kanten Nichtbaumkanten.

Da wir in jedem der einzelnen Verkleinerungsschritte das Innere um höchstens 1 verkleinern, folgt, dass nach endlich vielen Iterationen das Innere klein genug, und gleichzeitig das Äußere nicht zu groß geworden ist, und die Fundamentalkreissuche terminiert. Mit dem so gefundenen Fundamentalkreis lassen sich nun ebenfalls der Separator sowie die Mengen V_1 und V_2 zusammensetzen.

3.3.4 Zusammensetzen des Fundamentalkreisseperators

Der aus der Fundamentalkreissuche gefundene Kreis C führt nun auf dem ursprünglichen Graphen G direkt zu einem Separator. Wir wollen mit $V_1' = \text{Äußeres}(C)$ und $V_2' = \text{Inneres}(C)$ die Knoten bezeichnen, die im Äußeren bzw. Inneren des Fundamentalkreises C (in dem kontrahierten Graphen G') liegen. Nun stellen wir zunächst den Graphen G wieder her. Das heißt, wir löschen die Triangulationskanten, stellen die Knoten und Kanten aus den Levels 0 bis m, die wir verschmolzen haben, wieder her, und machen außerdem das Löschen der unteren Level M bis h rückgängig. Da wir beim Verschmelzen einen Keller für die Einfüge- und Löschoperationen benutzt haben, können wir diesen Keller abarbeiten und für jeden Schritt die inverse Operation durchführen. Die Mengen S, V_1 und V_2 auf G setzen sich nun wie folgt zusammen:

- $S := (C \cup S_m \cup S_M) \setminus s$. Dabei sind S_m und S_M die m- bzw. M-Level und s die Menge der Knoten, die verschmolzen wurden.
- V_1 ist die größere der beiden Mengen V_1' und V_2'. Ist $|V_1'| = |V_2'|$, so ist V_1 die Menge, die s nicht enthält.
- $V_2 = V \setminus (S \cup V_1)$.

Dies garantiert uns, dass sowohl $|V_1|$ als auch $|V_2|$ die Größe $2n/3$ nicht übersteigen.

Man erkennt, dass Phase III von der Laufzeit sicherlich die aufwändigste der drei Phasen ist. Die kniffligsten Stellen in der Implementierung stellten das Löschen der Level, vor allem das Wiederherstellen der Knoten und Kanten, sowie die Unterscheidung von Äußerem und Inneren bei einem Fundamentalkreis dar, worauf in den Beweisen des Theorems nicht eingegangen wird.

Die Qualität der Mengen V_1 und V_2, die von Phase III erzeugt werden unterscheidet sich in unserer Implementierung deutlich von der Qualität aus Phase I oder II. Erfüllt nämlich der erste generierte Fundamentalkreis nicht die gewünschten Eigenschaften (ist er beispielsweise zu groß), so wird er in jedem Iterationsschritt der Fundamentalkreissuche um genau einen Knoten im Inneren verkleinert, bis er gerade die Balancebedingung von Innerem und Äußerem erfüllt. Es ist leicht ersichtlich, dass die so gefundene Partition bezüglich V_1 und V_2 schlecht ausgewogen ist. Dies hat sich auch in unseren Experimenten bestätigt, auf die wir im übernächsten Abschnitt eingehen möchten. Zuvor möchten wir aber noch ein paar Dinge zur Visualisierung des Algorithmus vorstellen.

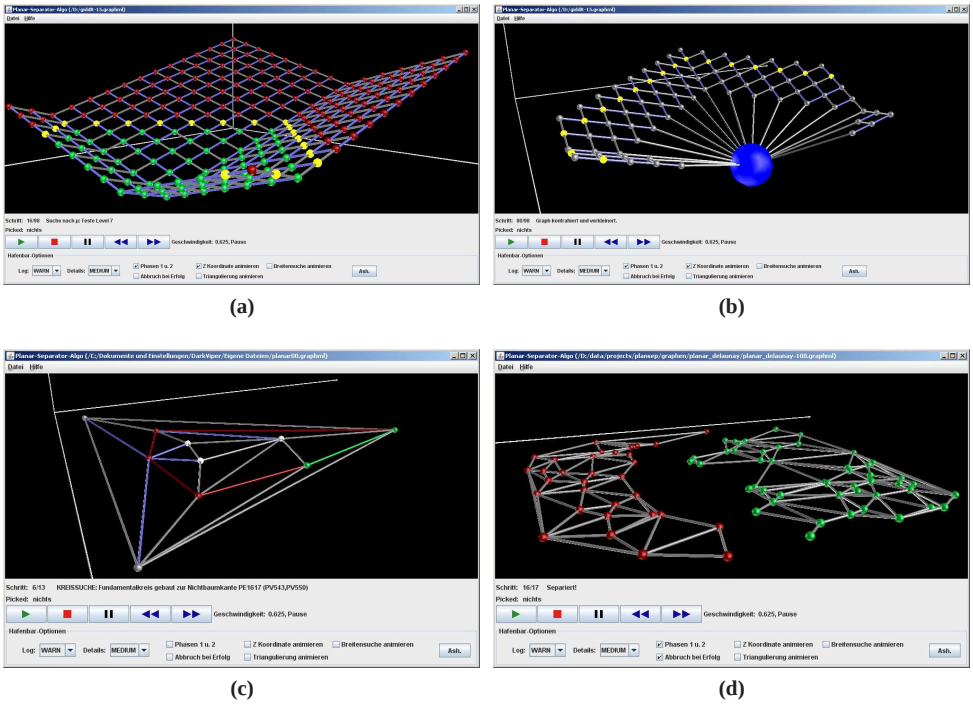
4 Visualisierung

Zur Speicherung der Graph-Datenstruktur mit allen graphenbezogenen Metadaten verwenden wir die JungX-Bibliothek [junb], die die Jung-Graphenbibliothek [juna] um planare Graphen erweitert. Das Algovis3D-Framework [alg] stellt den Graphen unter Benutzung von Java3D [jav] selbstständig in einer 3D-Umgebung in Form von Kugeln und Zylindern dar. Es bietet weiterhin die notwendigen Schnittstellen, um die Graphdarstellung im Algorithmenverlauf bequem zu verändern.

Da der Planar-Separator-Algorithmus in seiner Gesamtheit recht komplex ist, bemühten wir uns, für jeden seiner Ausführungsschritte eine Darstellung zu finden, die das Verständnis des aktuellen Vorgangs erleichtert.

Alle Schritte des Algorithmus verwenden Farbmarkierungen, um deutlich zu machen, welche Knoten oder Kanten in einem jeweiligen Ausführungsschritt betrachtet werden oder schon betrachtet worden sind. Darüberhinaus gibt es noch einige Operationen, die die Position oder Gestalt von Knoten verändern: Beim Aufbauen des Breitensuchbaums in Phase I wird der Level eines jeweiligen Knotens durch eine Verschiebung längs der Z-Achse dargestellt; je höher der Level eines Knoten ist, desto höher ist seine Z-Koordinate. In späteren Schritten werden die Knoten wieder auf dieselbe Höhe gebracht, um den Betrachter nicht unnötig zu verwirren. Beim Verschmelzen von Knoten bemühten wir uns um eine intuitive Darstellung des Vorgangs: Ein Knoten wird so in einen anderen „hineingeschmolzen“, dass er in seinen Vaterknoten hineinbewegt wird. Der Vaterknoten selbst wird so vergrößert, dass sein Volumen um das des Kindknotens wächst. Beim Triangulieren des Graphen erhalten die eingefügten Kantenzylinder einen geringerer Durchmesser, um Graph- und Triangulierungskanten optisch deutlich voneinander zu trennen.

Nachdem ein Separator gefunden ist, stellen wir den Graphen in seiner ursprünglichen Form



(a) **(b)**

(c) **(d)**

Abbildung 8: Schnappschüsse aus der Visualisierung des Algorithmus: (8a) Visualisierung der Levels des Breitensuchbaums durch die Höhe und Suche des μ-Levels in Phase I. (8b) Verschmelzungsschritt. (8c) Ein generierter Fundamentalkreis in Phase III. (8d) Ein separierter Graph nach Durchführung des Algorithmus.

wieder her und färben zunächst die Separatorknoten rot, um diese hervorzuheben. In einem Folgeschritt werden die Separatorknoten[8] gelöscht und die separierten Knotenmengen unterschiedlich eingefärbt, um das Endergebnis des Algorithmus ansprechend darzustellen.

Abbildung 8 zeigt einige Auszüge aus der Visualisierung des Algorithmus.

5 Experimente

Zusätzlich zur Implementierung und Visualisierung des Algorithmus möchten wir noch einige Experimente vorstellen, die wir am Planar-Separator-Algorithmus durchgeführt haben. Im Wesentlichen haben wir einige Graphklassen auf die Qualität der vom Algorithmus gefundenen Separatoren untersucht. Auf die Graphklassen und Parameter, die wir gemessen haben, gehen wir im nächsten Abschnitt ein. Danach folgt eine Auswertung und Interpretation der

[8] Inklusive ihrer inzidenten Kanten.

Ergebnisse. Da der Schwerpunkt des Praktikums in der Visualisierung des Algorithmus lag, gibt es durchaus noch weitere interessante Untersuchungen, die man durchführen könnte. Wir werden kurz am Ende dieses Kapitels einige Ideen für weitere Experimente präsentieren.

Für die Durchführung der Experimente entwickelten wir zusätzlich zur Visualisierungsumgebung eine weitere, die für Messungen ausgelegt ist. Sie nimmt über die Kommandozeile Parameter entgegen[9], und speichert Messdaten in einer Textdatei, so dass diese beispielsweise mit Gnuplot [gnu] ausgewertet werden können. Weiterhin verzichtet die Kommandozeilenumgebung auf die grafische Darstellung des Algorithmus.

5.1 Graphklassen und Messparameter

Die Grundlage für die Eingaben unserer Messungen war der Graph-Generator `gengraph` [gen] mit dem sich unterschiedliche Graphklassen generieren lassen, wobei man die Anzahl der Knoten und Kanten frei festlegen kann. Wir betrachten im Wesentlichen folgende drei Graphklassen:

- **Delaunay-Graphen.** Zu einer zufälligen Punktemenge im Einheitsrechteck wird das Voronoi-Diagramm berechnet. Der duale Graph bildet die Delaunay-Triangulierung. Um die Kantenzahl zu reduzieren, werden zufällig Kanten gelöscht.
- **Sechseckgitter.** Reguläre Graphen bestehend aus Sechsecken.
- **Dreiecke.** Reguläre Graphen bestehend aus Dreiecken. Diese sind, bis auf die äußere Facette, vollständig trianguliert.

In Abbildung 9 ist zu jeder Graphklasse eine Instanz abgebildet. Für jede der Klassen haben wir Graphen zwischen 10 und 20000 Knoten generiert, wobei die Anzahl generierter Graphen bei den Dreiecks- und Delaunay-Graphen 200, und bei den Sechseck-Gittern 100 beträgt. Alle Messungen sind somit in Abhängigkeit der Knotenanzahl n.

Gemessen haben wir das Folgende.

- **Phase des Erfolgs**. Welche Phase im Algorithmus führt als erste zum Erfolg?
- **Separatorgröße**. Uns interessierte hier die Größe des Separators. Theorem 1 garantiert eine Worst-Case-Separatorgröße von $4\sqrt{n}$. Um die tatsächliche Größe des Separators damit in Zusammenhang zu bringen und ein Maß zu erhalten wie gut der gefundene Separator (bezüglich der Größe) ist, haben wir die Ergebnisse auf den Term $4\sqrt{n}$ normiert und folgenden Zusammenhang gemessen:

$$n \mapsto \frac{|S|}{4\sqrt{n}}$$

- **Balance**. Zusätzlich zur Größe des Separators ist es wichtig, dass die Mengen V_1 und V_2 bezüglich ihrer Größe möglichst gut balanciert sind. Ein optimales Ergebnis liegt

[9] Somit können beispielsweise alle Graphen in einem angegebenen Verzeichnis mit dem Algorithmus getestet werden.

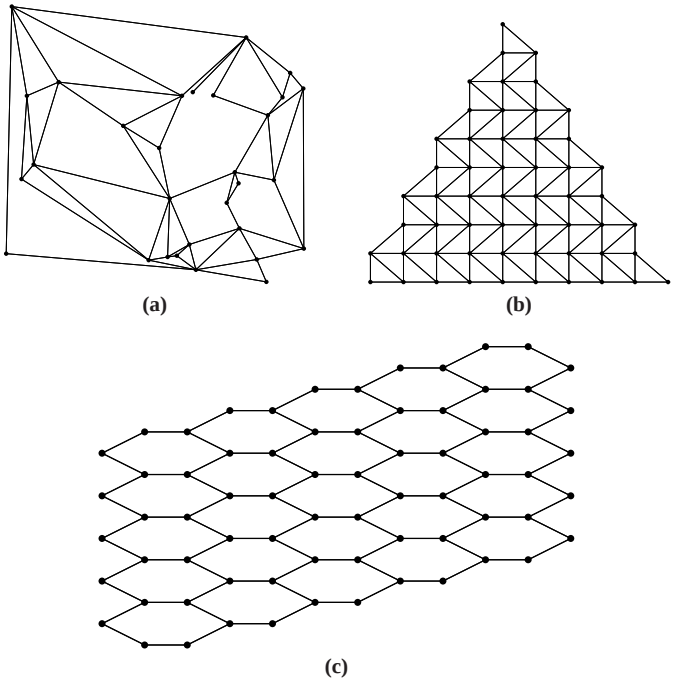


Abbildung 9: Betrachtete Klassen: (9a) Delaunay-Graphen, (9b) Dreiecksgraphen, (9c) Sechseck-Gitter.

vor, wenn $|V_1| = |V_2|$ gilt. Das Theorem 1 garantiert jedoch nur, dass sowohl V_1 als auch V_2 weniger als $2n/3$ Knoten enthalten. Sei o.B.d.A. $|V_1| \geq |V_2|$. Es gilt also im Optimalfall $|V_1|/|V_2| = 1$, während im Worst-Case

$$\frac{|V_1|}{|V_2|} = \frac{2n/3}{1n/3} = 2$$

gilt[10]. Um auch hier eine Güte für die Balance zu erhalten, haben wir uns für folgende Messung entschieden:

$$n \mapsto |1 - \frac{|V_1|}{|V_2|}|$$

Je näher das Ergebnis also an der 0 ist, desto besser die Balance.

- **Anzahl generierter Fundamentalkreise**. Sicherlich ist es auch für die Laufzeit des Algorithmus von Bedeutung, wie viele Fundamentalkreise in Phase III generiert werden müssen – wie oft also ein Verkleinerungs- bzw. Vergrößerungsschritt durchgeführt wird – bis ein separierender Kreis gefunden wird.

[10] Tatsächlich können auch Werte größer als 2 auftreten, falls $|V_1| = 2n/3$. In dem Fall ist $|S| + |V_2| = 1n/3$, und wegen $|S| > 0$ kann $|V_2| < 1n/3$ erfüllt sein.

Jede der Messungen haben wir auf allen drei Graphklassen durchgeführt. Wir haben außerdem die Ergebnisse betrachtet wenn man die Phasen I und II weglässt, das heißt die Fundamentalkreissuche direkt auf dem Originalgraphen durchführt. Die Theorie garantiert uns dann zwar die Schranke $4\sqrt{n}$ bezüglich der Separatorgröße nicht mehr, da die Höhe des Breitensuchbaums auf dem Graphen durchaus in $\mathcal{O}(n)$ liegt und somit die Größe eines Kreises auch $\mathcal{O}(n)$ Knoten beinhalten kann. Dennoch haben die experimentellen Resultate gezeigt, dass die Größe der Separatoren mit dieser Methode nicht wesentlich schlechter ausfällt. Die Messung bezüglich der generierten Fundamentalkreise haben wir auch nur unter Ausführung von Phase III gemessen, da bei Erfolg in Phase I bzw. II überhaupt keine Kreise generiert würden.

5.2 Auswertung

Die Messdaten im Folgenden haben teilweise eine recht hohe Streuung. Wir haben uns daher entschlossen, nicht die Messpunkte direkt anzugeben, sondern eine Bézier-Kurve durch die Daten zu legen. Dadurch sind die Tendenzen weitaus besser sichtbar. Die Streuung der Originaldaten kann dadurch erklärt werden, dass die Wahl der Wurzel des Breitensuchbaums, sowie der ersten Nichtbaumkante bei der Fundamentalkreissuche nicht deterministisch ablaufen, aber dennoch einen hohen Einfluss auf die Qualität des Ergebnisses haben können. Leider gibt es hier bislang keinen Ansatzpunkt, der immer zu einem besonders guten Ergebnis führt.

	Delaunay-Graphen	Dreiecke	Sechseck-Gitter
Phase I	111 (55%)	200 (100%)	100 (100%)
Phase II	0	0	0
Phase III	89 (45%)	0	0
Gesamt	200	200	100

Tabelle 1: Zum Erfolg führende Phase.

Tabelle 1 zeigt die Verteilung der zum Erfolg führenden Phasen bei den drei Graphklassen. Aufgrund der sehr regelmäßigen Struktur der Gitter- und Dreiecksgraphen sieht man, dass die erste Phase stets zum Erfolg führt, also der „mittlere“ Level μ im Breitensuchbaum für einen Separator hinreichend klein ist.

Abbildung 10 zeigt die Größe des Separators in Abhängigkeit der Anzahl der Knoten für jede der drei Graphklassen. Da wir die Größe auf $4\sqrt{n}$ normiert haben, bedeutet ein Wert von 1 die schlechtest mögliche Größe, während ein Wert nahe bei 0 einen sehr kleinen Separator angibt. Es ist deutlich zu erkennen, dass bei den Delaunay-Graphen, bedingt durch ihre recht zufällige Struktur, die Größe der Separatoren schlechter ausfällt als bei den regelmäßigen Gitter- und Dreiecksgraphen. Der Verlauf ist relativ konstant, was wenig überraschend ist.

Bezüglich der Balance in Abbildung 11 lässt sich Ähnliches sagen wie bei der Separatorgröße. Gegeben durch die sehr regelmäßige Struktur, führt die erste Phase bei den Dreiecks- und Gittergraphen bereits zu einer sehr ausgewogenen Partitionierung. Bei den Delaunay-Graphen ist das Verhalten chaotischer. Dennoch spricht ein Wert um 0.2 immernoch für eine sehr gute

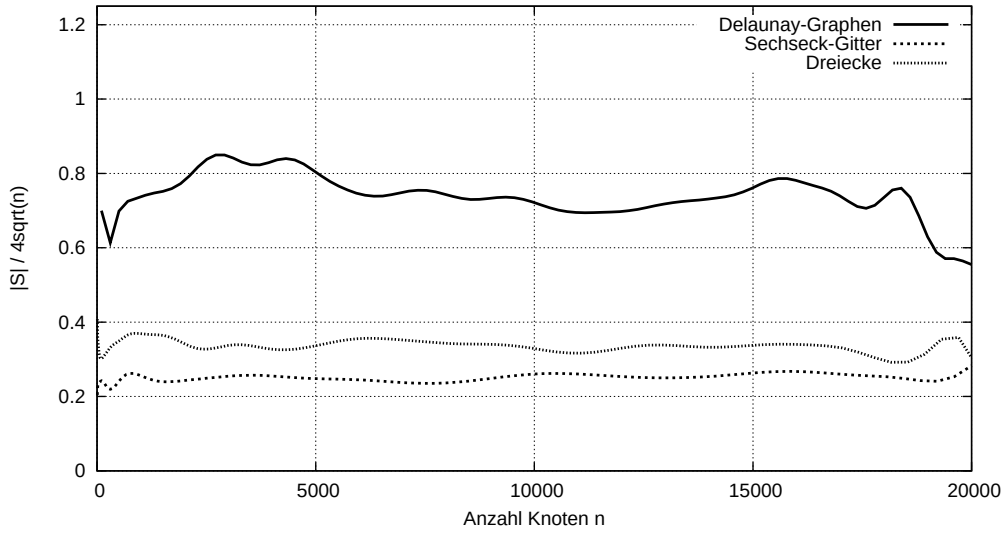


Abbildung 10: Relative Separatorgröße $|S|/4\sqrt{n}$.

Balance. Die Spitzen am Anfang der Kurven lassen sich dadurch erklären, dass bei Graphen mit sehr wenigen Knoten die Separatorknoten die Balance noch weitaus stärker beeinflussen.

Die nun folgenden Messungen beziehen sich ausschließlich auf Phase III. Das heißt wir berechnen lediglich einen Breitensuchbaum und führen dann die Fundamentalkreissuche direkt auf dem Eingabegraphen aus. Abbildung 12 zeigt die Separatorgröße mit dieser Methode. Wir haben die Kurve wieder auf $4\sqrt{n}$ normiert. Obwohl man erwarten würde, dass hier auch Separatoren mit mehr als $4\sqrt{n}$ Knoten auftreten, sind die Separatoren weit von der oberen Schranke entfernt. Dieses Resultat ist insoweit erstaunlich, da vor allem bei Delaunay-Graphen mit dieser Methode sogar wesentlich bessere Resultate erzielt werden als bei den Phase-I-Separatoren.

Der positive Effekt auf die Separatorgröße scheint sich aber in der Balance wieder auszugleichen. Wie man in Abbildung 13 sieht, ist die Balance der Mengen V_1 und V_2 weitaus schlechter als bei Durchführung aller Phasen. Besonders bei den regelmäßigen Graphen macht sich das bemerkbar, wo die Phase I extrem gut balancierte Mengen geliefert hat. Die schlechte Balance lässt sich dadurch erklären, dass bei der Fundamentalkreissuche in unserer Implementierung sofort der erste Kreis als Separator genommen wird, der den geforderten Balancebedingungen genügt. Dies führt natürlich zu einer Worst-Case-Balance in Phase III.

Abbildung 14 zeigt schließlich die Anzahl generierter Fundamentalkreise pro Graph mit n Knoten. Um den Zusammenhang zur Anzahl Knoten besser darzustellen, ist nicht die absolute Anzahl an Kreisen aufgetragen, sondern die Anzahl der Kreise pro 100 Graphknoten. Erstaunlicherweise liegt hier ein sublinearer Zusammenhang vor, denn mit zunehmender Größe des Graphen scheint die relative Anzahl an generierten Kreisen abzunehmen, und zwar bei

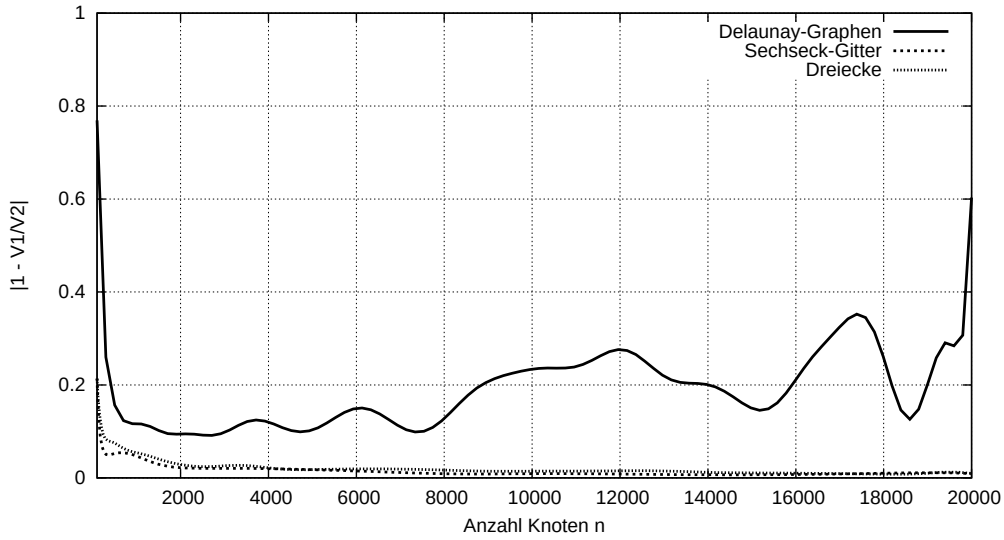


Abbildung 11: Balance der Mengen V_1 und V_2.

allen drei Graphklassen. Der Algorithmus wird also mit wachsender Graphgröße zunehmend effizienter.

Auch wenn die Messungen auf generierten Graphklassen beruhen, und in der Praxis möglicherweise strukturell andere Graphen eingesetzt werden, lassen sich ein paar Ergebnisse einsehen. Das erstaunlichste Resultat ist, dass die Phase-III-Separatoren auf dem Ursprungsgraphen durchaus gute Ergebnisse liefern. Die Separatorgröße scheint sogar bei unregelmäßigen Graphen besser auszufallen als bei Phase I. Leider geht dies zu Lasten der Balance, was sich aber durchaus noch optimieren lässt, indem man beispielsweise nicht den ersten passenden Fundamentalkreis als Separatorkandidaten wählt, sondern den Verkleinerungs- resp. Vergrößerungsschritt noch weiterlaufen lässt.

An dieser Stelle ist auch noch Platz für weitere Messungen. Es wäre auch interessant zu untersuchen, wie sich Separatorgröße und Balance in Bezug auf die Wahl der Breitensuchbaum-Wurzel bzw. der ersten Nichtbaumkante in der Fundamentalkreissuche verhalten. Die Laufzeit wäre ein weiterer interessanter Aspekt. Wir haben keinerlei Laufzeitmessungen durchgeführt, da im Algorithmus sehr viel Overhead für die Visualisierung erzeugt wird. Vorallem der enorme Speicherbedarf von Java bei steigender Graphkomplexität hätte uns die Ergebnisse verfälscht, da durch das Auslagern auf die Festplatte die Geschwindigkeit überproportional einbrechen würde. Dennoch wäre eine effiziente Implementierung des Algorithmus – beispielsweise in `C++` – mit einer schnellen Graphenbibliothek sicherlich für Laufzeitmessungen interessant.

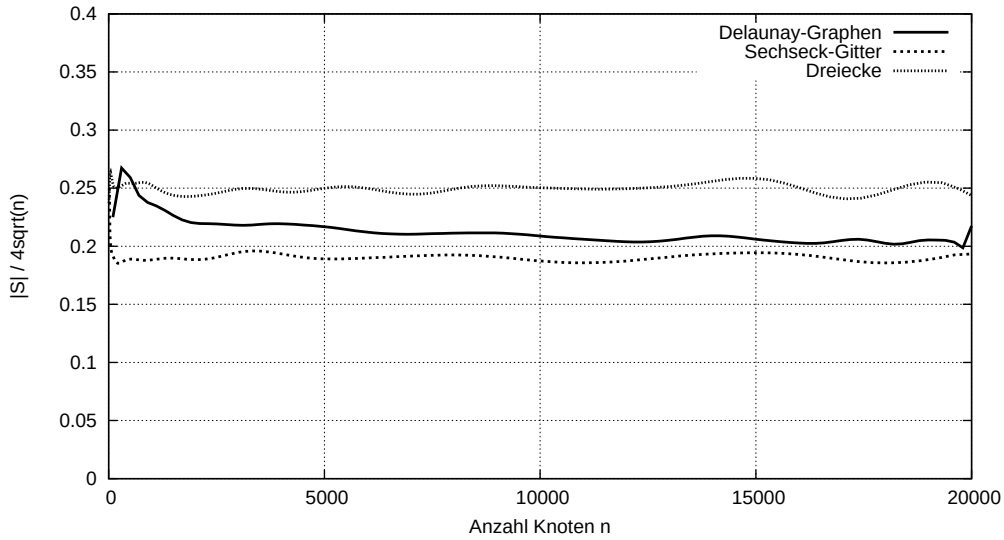


Abbildung 12: Relative Separatorgröße $|S|/4\sqrt{n}$ nur unter Betrachtung von Phase III.

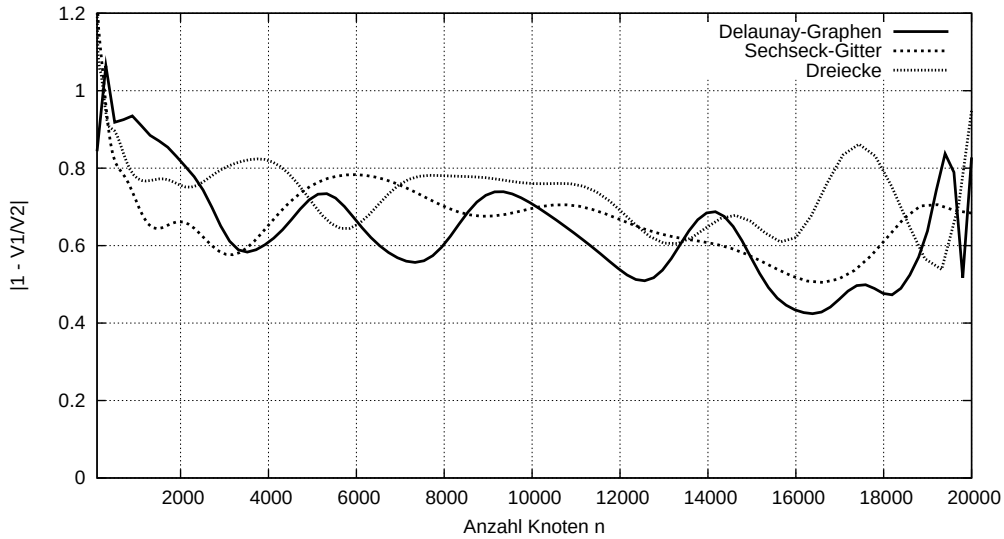


Abbildung 13: Balance der Mengen V_1 und V_2 nur unter Betrachtung von Phase III.

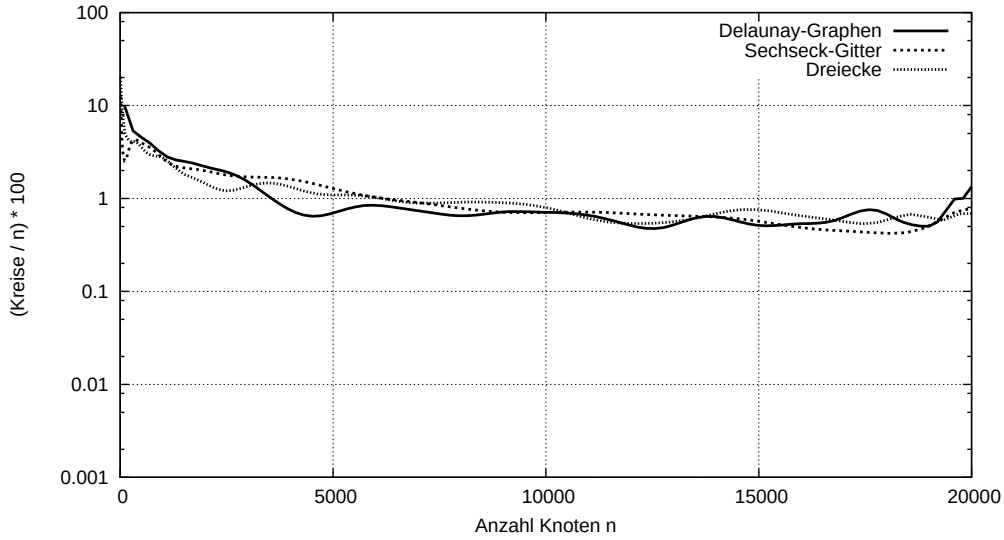


Abbildung 14: Anzahl generierter Fundamentalkreise.

6 Fazit

Unsere Implementierung des Planar-Separator-Theorems in Java zeigt die Möglichkeiten des Frameworks AlgoVis3D [alg] zur Visualisierung von Graphenalgorithmen auf. Eingesetzt in Lehrveranstaltungen wie Vorlesungen oder Seminaren, kann dies sehr zum Verständnis des Planar-Separator-Algorithmus beitragen.

Die Schwierigkeiten, die bei der Implementierung aufgetreten sind konnten wir alle mit Workarounds lösen, ohne dabei die Laufzeit des Algorithmus asymptotisch zu verschlechtern. Die Experimente konnten einen kurzen Überblick über die Qualität des Algorithmus liefern, wobei das wichtigste Resultat die Erkenntnis, dass die Fundamentalkreissuche für sich sehr gute Separatoren liefert, war. Eine weitergehende Optimierung bezüglich der Balance, indem man zum Beispiel nicht den ersten passenden Fundamentalkreis als Separator betrachtet, könnte die Qualität hier noch weiter steigern.

Weitere Erweiterungen am Projekt könnten zum Beispiel eine manuelle Auswahl der Breitensuchbaum-Wurzel oder der ersten Nichtbaumkante über das Visualisierungs-Framework beinhalten. Außerdem wäre es für die Visualisierung interessant, den Algorithmus auf dreidimensionalen Graphen, wie zum Beispiel Kugeln oder konvexen Polyedern, durchführen zu können – zumal das Visualisierungs-Framework auf Java3D aufbaut, und damit optimale Voraussetzungen hat. Weitere Experimente, insbesondere die Separatorgrößen und Balancen unter Wahl der Wurzel des Breitensuchbaums und der Nichtbaumkante, wären aufschlussreich und würden vielleicht zu einer guten Heuristik für die Berechnung dieser führen.

Literatur

[alg] *AlgoVis3D Visualisierungsframework*. http://i11www.iti.uni-karlsruhe.de/teaching/WS_0607/planalgo/AlgoVis3D/doc/api/index.html

[Die06] DIESTEL, Reinhradt: *Graphentheorie*. 3. Berlin : Springer, 2006. – ISBN 3–540–21391–0

[gen] *Graph-Generator*. http://i11www.iti.uni-karlsruhe.de/resources/graphgenerator.php

[gnu] *Gnuplot*. http://www.gnuplot.info/

[jav] *Java 3D API*. https://java3d.dev.java.net/

[juna] *JUNG Java Universal Network/Graph Framework*. http://jung.sourceforge.net

[junb] *JUNGX Graphenbibliothek*. http://i11www.iti.uni-karlsruhe.de/teaching/WS_0607/planalgo/jungX/doc/index.html

[LT77] LIPTON, Richard J. ; TARJAN, Robert E.: A separator theorem for planar graphs. Stanford, CA, USA : Stanford University, 1977. – Forschungsbericht

[Paj07] PAJOR, Thomas: *Triangulierung eines Planaren Graphen*. Februar 2007. – Beschreibung eines $O(n)$ Algorithmus zur Triangulierung eines planaren Graphen

[Wag06] WAGNER, Prof. Dr. D.: Algorithmen auf planaren Graphen. (2006)