Planar-Separator-Theorem

Ausarbeitung zum Praktikum
Zerlegen und Clustern von Graphen
im SS 07

Betreuer: Martin Holzer

Hilal Akbaba, Stefan Hartte

3. November 2007

Zusammenfassung

Diese Ausarbeitung zum Praktikum „Zerlegen und Clustern von Graphen“ handelt von dem Planar-Separator-Theorem, das einen effizienten Algorithmus zum Zerlegen eines planaren Graphen in $O(n)$ in unzusammenhängende Teilgraphen durch einen Separator[1] liefert. Dabei kann garantiert werden, dass die Teilgraphen höchstens $\frac{2}{3} \cdot n$ Knoten haben und der Separator höchstens $4 \cdot \sqrt{n}$ Knoten hat. Wir haben den Algorithmus mit Hilfe von JUNG [juna] in Java [jav] implementiert. Im Weiteren haben wir noch Experimente bzgl. der Separatorgröße mit dem Algorithmus auf verschiedenen Graphenarten durchgeführt, um dessen Verhalten zu analysieren. Dabei war die Separatorgröße in etwa gleichbleibend für Delaunay Graphen (zwischen 40 und 55% der maximal möglichen Separatorgröße).

1 Einleitung

In vielen Anwendungsgebieten wie bei Straßennetzen, bei Netzwerk- und Routenplanungen, aber auch in vielen weiteren Anwendungsbereichen werden planare Graphen eingesetzt. Da sie von großem Interesse sind, ist es wichtig, effiziente Algorithmen für planare Graphen zu finden.

Oft wird bei Algorithmen das Prinzip von Teile und Herrsche[2] angewendet, wobei der Graph erstmal in mehrere Teilgraphen zerlegt werden muss. Hier hilft uns das Planar-Seperator-Theorem, das besagt, dass ein planarer Graph in linearer Laufzeit in Teilgraphen aufgeteilt werden kann, wobei den Knotenmengen eine maximale Größe von $\frac{2}{3} \cdot n$ zugesichert wird. Dabei bezeichnet n die Anzahl der Knoten des ursprünglichen Graphen. Dies ist wichtig, da die Größe der Teilprobleme die Laufzeit bestimmt.

[1] Ein Separator ist eine Knotenmenge eines Graphen, der durch Wegnahme dieser Menge in mindestens zwei unzusammenhängende Teilgraphen zerfällt.

[2] Das Prinzip besagt, dass ein Problem dadurch gelöst wird, das man es in Teilprobleme solange zerlegt, bis diese so klein sind, dass sie direkt effizient gelöst werden können. Danach wird die Lösung der Teilprobleme wieder zusammengesetzt zu einer Gesamtlösung.

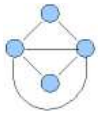
Das Planar-Separator-Theorem von Lipton & Tarjan (1977) [LT77] ist konstruktiv, d. h. es liefert einen Algorithmus, den wir in unserem Praktikum in JAVA 1.5 [jav] implementiert haben. Hierfür verwenden wir JUNG-API [juna], dies ist eine frei verfügbare Graphenbibliothek, die in der Programmiersprache JAVA geschrieben ist. Zusätzlich benutzen wir JUNGX [junb], eine Erweiterung von JUNG, die wir insbesondere für planare Graphen benötigen. Speziell bei den Experimenten nahmen wir den Grapheditor yEd [yEd] sowie den Graphgen [3] zu Hilfe. Um die theoretischen Grundlagen zu erarbeiten, recherchierten wir im Rahmen des Praktikums das Skript „Algorithmen für planare Graphen“ [Wag06], sowie den technischen Bericht „Triangulierung eines planaren Graphen“ [Paj07]. Auf die Hilfsmittel, die wir für die Experimente benötigen, werden wir auch in Abschnitt 5.1 eingehen.

Im Folgenden möchten wir die Definitionen erläutern, das grundlegende Theorem verdeutlichen und danach auf den Algorithmus eingehen. Anschließend gehen wir näher auf die Implementierung, die Visualisierung und die damit verbundenen Hindernisse ein. Um unseren Algorithmus zu testen, führten wir Experimente bzgl. der Separatorgröße durch. Letztendlich geben wir noch ein Ausblick auf weitere interessante Fragestellungen und Verbessungsmöglichkeiten z. B. der Visualisierung, der Wahl der Wurzel und der Wahl der Nichtbaumkante.

2 Definitionen

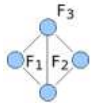
Im Folgenden wollen wir auf die wichtigsten Definitionen, die uns noch mehrmals begegnen werden, eingehen und diese anhand von Beispielen verdeutlichen.

Ein **planarer Graph** ist ein Graph, den man kreuzungsfrei in der Ebene zeichnen kann. In der Skizze ist ein planarer Graph zu sehen, wo keine zwei Kanten sich in der Ebene kreuzen.



Eine **Triangulierung** ist ein kantenmaximaler planarer Graph, der aus einem planaren Graphen durch Hinzufügen von Kanten entsteht. Dieser Vorgang wird auch Triangulierung genannt. Die Triangulierung des obersten Skizze wird darunter veranschaulicht.

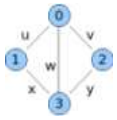
Eine zu einem Knoten **inzidente** Kante bedeutet, dass die Kante mit dem Knoten verbunden ist. Im Beispiel ist die Kante u bzw. v inzident zu dem Knoten 0. Genauso nennt man zwei Kanten **inzident**, wenn sie mit einem gemeinsamen Knoten verbunden sind. Da die Kanten u und v mit dem gemeinsamen Knoten 0 verbunden sind, nennt man sie auch inzidente Kanten.



Eine **Facette** ist ein Gebiet, das durch die Kanten begrenzt ist, aber nicht durch eine Kante geteilt wird. Wie im Schaubild zu sehen ist, gibt es innere, begrenzte Facetten (F_1,F_2) und auch eine äußere, unbegrenzte Facette (F_3).

[3] Ein Graphengenerator der Lehrstuhls

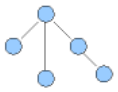
Ein **Weg** in einem Graphen besteht aus einer Kantenfolge, in dem alle Kanten miteinander verbunden sind. Hier gibt es viele mögliche Wege. Zum Beispiel vom Knoten 1 zum Knoten 2 über die Kanten u, v oder x, w, v usw...



Ein **Zyklus** in einem Graph ist ein Weg, dessen Anfangs- und Endknoten identisch sind. Hier kann man bei dem gleichen Bild ein Zyklus z. B. u, v, y, x oder auch u, w, x finden.

Ein **Kreis** ist ein Zyklus, wobei mit jeden Knoten eine gerade Anzahl von Kanten verbunden ist. Das Beispiel zeigt einen möglichen Kreis mit der Kantenmenge $\{u,\ v,\ y,\ x\}$. Bei jedem Knoten sind je 2 (gerade Anzahl) Kanten inzident, z. B. zu Knoten 2 die Kanten v und y.

In einem **zusammenhängenden** Graphen gibt es von jedem Knoten aus jeweils einen Weg zu allen anderen Knoten.

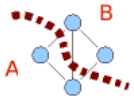


Ein **Baum** ist ein zusammenhängender Graph, in dem es keinen Zyklus gibt.

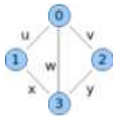
Eine **geometrische Einbettung** ist eine Darstellung eines Graphen, bei der die Koordinaten der Knoten und seine dazu inzidenten Kanten in beliebiger Reihenfolge durch die Knotenpaare angegeben sind.

Eine **kombinatorische Einbettung** ist eine Speicherung eines Graphen, bei der jeweils die Reihenfolge der zu einem Knoten inzidenten Kanten angegeben ist. Die Reihenfolge verläuft meist Gegenuhrzeigersinn oder in Uhrzeigersinn.

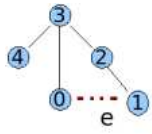
Eine **Partition** besteht aus mehreren Mengen, die disjunkt[4] sind und vereinigt die Gesammtmenge ergeben. Bei uns bestehen die Mengen aus Knoten. Im Beispiel besteht die Partition aus zwei Mengen A und B. Die Mengen sind durch die gestrichelte Linie getrennt. Es könnte aber auch mehr als zwei Mengen geben.



Ein **Separator** ist eine Knotenmenge eines Graphen, der durch Wegnahme dieser Menge in mindestens zwei unzusammenhängende Graphen zerfällt. Im Schaubild wäre die Knotenmenge $\{0, 3\}$ ein möglicher Separator. Dabei entstehen zwei unzusammenhängende Teilgraphen, nämlich mit dem Knotenmengen $\{1\}$ und $\{4\}$.



Eine **Nichtbaumkante** ist eine Kante in einem gegebenen Graphen, wobei diese Kante nicht zu dem Baum desselben Graphen gehört. Die Kante e ist Nichtbaumkante bzgl. des Baumes, der aus allen Kanten bis auf e besteht.



Ein **Fundamentalkreis** ist ein Kreis, der durch eine Nichtbaumkante und einen Weg in einem Baum gebildet wird. Dieser Weg beginnt bei dem einen Knoten der Nichtbaumkante und endet bei dem anderen Knoten der Nichtbaumkante. In unserem Beispiel wird durch die Nichtbaumkante e ein Kreis realisiert, der aus allen Kanten bis auf die Kante zwischen dem Knoten 3 und 4 besteht, da diese Kante nicht zum Weg im Baum gehört.

[4] Es gibt keine gleichen Elemente in je zwei verschiedenen Mengen.

Ein **Level eines Knotens** ist die Anzahl der Kanten des kürzesten Weges von dem Knoten zur Wurzel des Baumes.

3 Algorithmus

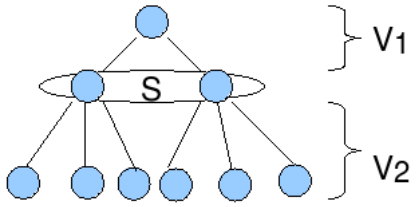
Wir werden im Folgenden das Prinzip des Algorithmus, der auf dem Planar-Separator-Theorem basiert, verdeutlichen. Wer sich für den Beweis des Algorithmus interessiert, der sei auf das Skript [Wag06] verwiesen. In jeder Phase wird ein Seperator gesucht, falls dieser nicht geeignet ist, wird zur nächsten Phase übergegangen. Doch vorerst einmal erläutern wir das grundlegende Planar-Separator-Theorem.

3.1 Planar-Separator-Theorem

Die Knotenmenge eines zusammenhängenden, planaren Graphen $G = (V, E)$, n = $|V| \geq 5$, kann so in drei Mengen V1 , V2 , S $\subseteq$ V partitioniert werden, dass

1. $|V1|, |V2| \leq \frac{2}{3} \cdot n$,
2. S Separator, der V1 und V2 trennt,
3. $|S| \leq 4 \cdot \sqrt{n}$.

Diese Partition kann in der Laufzeit $O(n)$ berechnet werden.



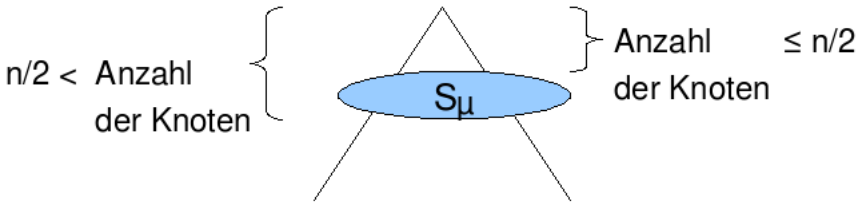
Das obige Bild zeigt einen Graphen, der durch den Separator S in zwei unzusammenhängende Teilgraphen (V_1,V_2) zerlegt. Der Graph, der nicht zwangsläufig ein Baum sein muß, sondern ein beliebiger, zusammenhängender, planarer Graph, wie im Planar-Separator-Theorem gefordert.

3.2 1. Phase

In der ersten Phase des Algorithmus wird von einem frei gewählten Wurzelknoten ein Baum mit Breitensuche[5] aufgebaut. Anschließend wird ein „mittleres Level“ S_μ gesucht, wobei wir die Level von oben nach unten mit $S_0, S_1, S_2, ...$ bezeichnen und gilt:

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

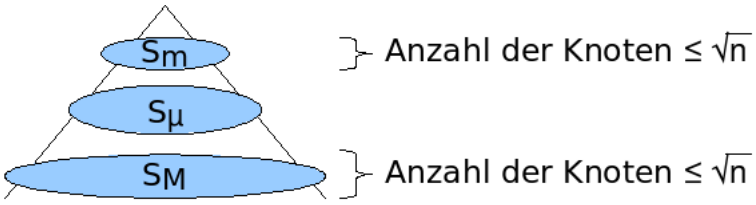
[5] Suche, bei der die direkten Nachfolgerknoten zuerst besucht werden.



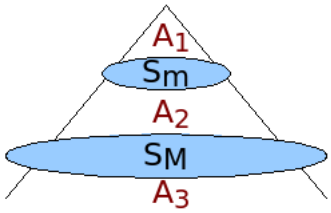
Falls S_μ weniger als $4 \cdot \sqrt{n}$ Knoten hat, sind alle Bedingungen erfüllt, der Algorithmus liefert S_μ als einen geeigneten Separator. Die Mengen V_1 und V_2 sind dann oberhalb und unterhalb des Separator S_μ.

3.3 2. Phase

In der zweiten Phase suchen wir zwei weitere Levels S_m und S_M, wobei S_m oberhalb von S_μ, S_M unterhalb von S_μ gelegen sind so, dass diese gleich oder weniger als $\sqrt{n}$ Knoten haben.



Man prüft, ob die Anzahl der Knoten zwischen S_m und S_M weniger als $\frac{2}{3} \cdot n$ Knoten vorhanden sind. Falls dies der Fall ist, haben wir bereits in der zweiten Phase einen geeigneten Separator gefunden. Nun müssen wir noch die Mengen V_1 und V_2 bestimmen. Dazu definieren wir drei Mengen A_1,A_2 und A_3. A_1 oberhalb von S_m, A_2 zwischen S_m und S_M und A_3 unterhalb von S_M.



Die Menge A_1 hat höchstens $\frac{n}{2}$ Knoten, da sie oberhalb von S_m, also auch oberhalb von S_μ liegt. Analoges gilt auch für A_3, das weniger als $\frac{n}{2}$ Knoten hat und A_2, das weniger als $\frac{2}{3} \cdot n$ Knoten besitzt (dies hatten wir bereits geprüft). Man wählt als V_1 die größte Menge aus A_1, A_2 und A_3, dann wissen wir bereits, dass diese Menge maximal $\frac{2}{3} \cdot n$ Knoten haben kann (wegen A_2). V_2 besteht aus den restlichen Knoten ohne V_1 und ohne S, wodurch unsere zweite Ungleichung $V_2 \leq n - V_1$ Zustande kommt. Dazu muss begründet werden, dass die Menge V_2 aus höchstens $\frac{2}{3} \cdot n$ Knoten besteht, wie im Planar-Separator-Theorem gefordert. Die Überlegung stammt aus den bekannten Voraussetzungen, woraus die erste

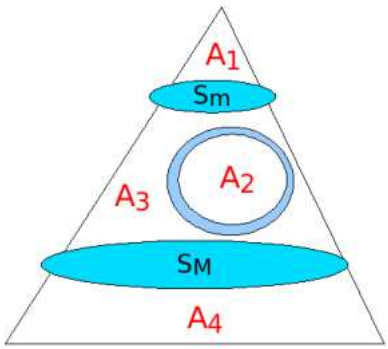
Ungleichung $V_2 \leq V_1 \cdot 2$ ableitbar ist. Es sei $V_1 = \max\{A_1, A_2, A_3\}$, d. h. V_1 kleiner gleich A_1, A_2 und auch als A_3, also gilt dies auch für die Summe von Beiden, was V_2 entspricht. Man kann in die zweite Ungleichung $V_2 \leq n - V_1$ die erste Ungleichung einsetzen, damit folgt $V_2 \leq n - 2 \cdot V_2$. Somit erhalten wir die erwünschte Bedingung auch erfüllt: $V_2 \leq \frac{2}{3} \cdot n$.

Die Separatorgröße ist auch eingehalten, da S_m und S_M jeweils höchstens $\sqrt{n}$ Knoten haben, also zusammen $2 \cdot \sqrt{n}$, obwohl nur $4 \cdot \sqrt{n}$ vom Theorem gefordert wurde.

3.4 3. Phase

Falls die Level S_m und S_M keinen geeigneten Separator bilden, kopiert man den Graphen. Man verändert den kopierten Graphen, sucht aber dennoch im ursprünglichen Graphen einen Separator. Im Anschluss findet man in dem veränderten Graphen einen Fundamentalkreis, der zusammen mit S_m und S_M einen Separator im ursprünglichen Graphen bildet.

Im Folgendem verschmilzt man alle Level oberhalb von S_m sowie S_m zu einem Knoten und löscht alle Level unterhalb von S_M sowie S_M (falls vorhanden). Der neu entstandene Graph wird trianguliert. Wir können dann einen beliebigen Fundamentalkreis wählen. Der endgültige Fundamentalkreis wird gesucht, indem wir diesen Kreis entsprechend vergrößern bzw. verkleinern, je nach Anzahl der Knoten, die sich innerhalb des Kreises befinden. Die Verkleinerung bzw. Vergrößerung werden wir später bei der Implementierung noch genauer betrachten. Wichtig ist jedoch, dass so lange vergrößert wird, bis mehr als $\frac{n}{3}$ Knoten bzw. verkleinert bis weniger als $\frac{2}{3} \cdot n$ Knoten im Kreis vorhanden sind, damit die geforderten Eigenschaften des Planar-Separator-Theorems erfüllt werden. Im Folgenden verdeutlichen wir uns noch, wie sich die Mengen V_1 und V_2 zusammensetzen. Hierzu schauen wir uns an, welche Mengen vorhanden sind.



Es existieren folgende Mengen: A_1 oberhalb von S_m, A_2 innerhalb des Kreises, A_3 zwischen S_m und S_M außerhalb des Kreises und A_4 unterhalb von S_M. Die Menge A_2 ist nach Kreiskonstruktion größer als $\frac{n}{3}$ und kleiner als $\frac{2}{3} \cdot n$. Also kann A_3 höchstens $\frac{2}{3} \cdot n$ groß sein, da A_2 bereits $\frac{n}{3}$ groß sein muss. V_1 besteht aus der größeren Menge entweder aus A_2 oder aus A_3. Diese Menge ist größer als $\frac{n}{3}$ und kleiner als $\frac{2}{3} \cdot n$. Deshalb muss V_2 als der Rest kleiner als $\frac{2}{3} \cdot n$ sein, weil V_1 größer als $\frac{n}{3}$ ist.

Nachfolgend wollen wir noch die Separatorgröße untersuchen. Wie wir schon aus Phase 2 wissen, sind S_m und S_M jeweils höchstens $\sqrt{n}$ Knoten groß. Also betrachten wir den Kreis. Hierzu benötigen wir eine Abschätzung über die Höhe des Baumes des veränderten Graphen, der zwischen den Level S_m und S_M liegt. Nach Konstruktion von S_m und S_M wissen wir bereits, dass diese Level aus mehr als $\sqrt{n}$ Knoten bestehen, da sie sonst als S_m bzw. S_M gewählt worden wären. Dann kann die Anzahl der Level maximal $\sqrt{n}$ betragen, d. h. aber für den Kreis, dass er aus maximal $2 \cdot \sqrt{n} + 1$ Knoten besteht. Falls der Kreis aber aus $2 \cdot \sqrt{n} + 1$ Knoten besteht, beinhaltet er die Wurzel, die als Knoten in der Separatormenge nicht notwendig ist, da S_m den Kreis bereits abschließt. Daraus ergibt sich eine Gesamtgröße des Separators aus Level S_m, S_M und dem Kreis (ohne Wurzel) von höchstens $\sqrt{n} + \sqrt{n} + 2 \cdot \sqrt{n} = 4 \cdot \sqrt{n}$.

4 Implementierung

Bei der Implementierung haben wir die drei Phasen des Algorithmus, der auf dem Planar-Separator-Theorem beruht, realisiert. Hierbei ergaben sich Schwierigkeiten, die meist erst bei der Implementierung aufgefallen sind. Insbesondere die Visualisierung der Triangulierung erwies sich als schwierig. Darauf werden wir im Folgenden eingehen.

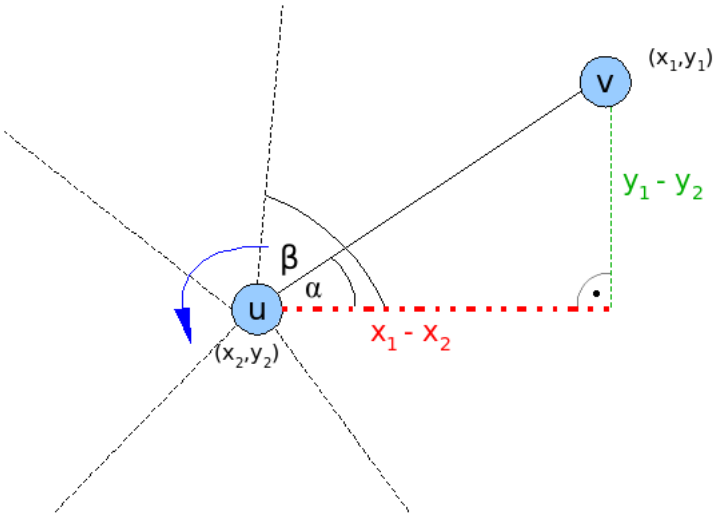
4.1 Vorarbeit

Zuerst haben wir einen Graphen im GraphML-Format eingelesen. Dazu verwendeten wir eine Bibliotek[6] des Lehrstuhls. Wir haben den Graphen, der in der geometrischen Einbettung gegeben war, in die kombinatorische Einbettung umgewandelt. Im Praktikum entschieden wir uns für die Reihenfolge Gegenuhrzeigersinn bei der kombinatorischen Einbettung.

Von jedem Knoten wird der Winkel von einer waagrechten Geraden (unten im Bild mit rotgestrichelten Linie) nach rechts zu allen seinen inzidenten Kanten ausgerechnet. Durch die Längen der Strecken $y_1 - y_2$ und $x_1 - x_2$ können wir mit Hilfe von $arctan(\frac{y_1 - y_2}{x_1 - x_2})$ den Winkel α berechnen. Analog werden die anderen Winkel ausgerechnet. Dabei ist zu beachten, dass in jedem Quadranten die Winkel anders zu berechnen sind. Die Liste der Winkel ($\{\alpha, \beta, ...\}$) haben wir dann mit Mergesort[7] sortiert. Somit erhalten wir eine Liste, die die im Gegenuhrzeigersinn sortierten Winkel der Kanten enthält. Mit Hilfe dieser Liste erhalten wir nun die Reihenfolge der Kanten für die kombinatorische Einbettung.

[6] Ein Mitarbeiter hatte uns diese freundlicherweise zur Verfügung gestellt

[7] Ein Sortieralgorithmus, der das Prinzip von Teile und Herrsche anwendet.



4.2 1. Phase

Die erste Phase erweist sich als relativ einfach. Das wesentliche Hilfsmittel besteht aus einem aufspannenden Baum[8]. Dieser wird unter Zuhilfenahme einer Breitensuche gefunden. Hierzu orientieren wir uns im Wesentlichen an [CLRS01]. Nach der Breitensuche erhalten wir durch die Baumstruktur einzelne Level, in welchen wir das „mittlere“ Level suchen.

Die **Breitensuche** baut einen aufspannenden Baum zu einem gegebenen Wurzelknoten auf, indem man eine Warteschlange[9] benutzt. Dies erfüllt den Zweck, dass die Knoten, die zuerst entdeckt werden auch zuerst verfolgt werden, d. h. dass alle Knoten eines Levels erst abgearbeitet werden, bevor die Knoten des nächsten Levels bearbeitet werden. Jeder Knoten kann sich hierbei in drei verschiedenen Zuständen befinden:

1. Der Knoten wurde noch nicht besucht.
2. Der Knoten wird besucht, es sind aber nicht alle seine Nachfolgerknoten besucht worden.
3. Der Knoten und alle seine Nachfolgerknoten sind besucht worden.

Diese Zustände sind notwendig, damit alle Knoten bearbeitet werden, alle Nachfolgerknoten entdeckt werden und jeder Knoten höchstens einmal bearbeitet wird. Das Level eines Knotens ist das Level seines Vorgängerknotens um Eins erhöht. Desweiteren haben wir uns die Reihenfolge, in der die Knoten besucht wurden, gespeichert, da wir diese bei der Zählung der Knoten, die innerhalb bzw. außerhalb des Kreises liegen, benötigen.

[8] Ein Baum, der maximal ist, d. h. dass der Baum sich nicht mehr erweitern lässt, ohne dass ein Zyklus entsteht.

[9] Eine Schlange ist eine Datenstruktur, bei der das zuerst hinzugefügte Element auch zuerst wieder entnommen wird.

Das **mittlere Level** wird im Weiteren gesucht. Wir können zählen, wieviel Knoten sich in jeweiligen Level befinden. Wie in Abschnitt 3.2 schon betrachtet, bestimmen wir das mittlere Level über die Summe der Knoten in den Leveln. Es wird dann überprüft, ob das mittlere Level bereits ein geeigneter Separator darsteltt. Dies ist der Fall, wenn die Anzahl der Knoten des mittleren Levels kleiner gleich $4 \cdot \sqrt{n}$ ist. Anderenfalls wird in Phase 2 übergegangen.

4.3 2. Phase

Da die Separatorgröße in der ersten Phase mehr als $4 \cdot \sqrt{n}$ ist, suchen wir die Level S_m und S_M, wie in Abschnitt 3.3 erläutert.

Um **einen geeigneten Separator** zu finden, addieren wir die Knoten der Level zwischen S_m und S_M. Die Anzahl der Knoten in den Leveln wurden bereits in der ersten Phase bestimmt. Falls diese Summe weniger als $\frac{2}{3} \cdot n$ ist, haben wir einen geeigneten Separator gefunden, ansonsten finden wir auf jeden Fall in Phase 3 einen Separator.

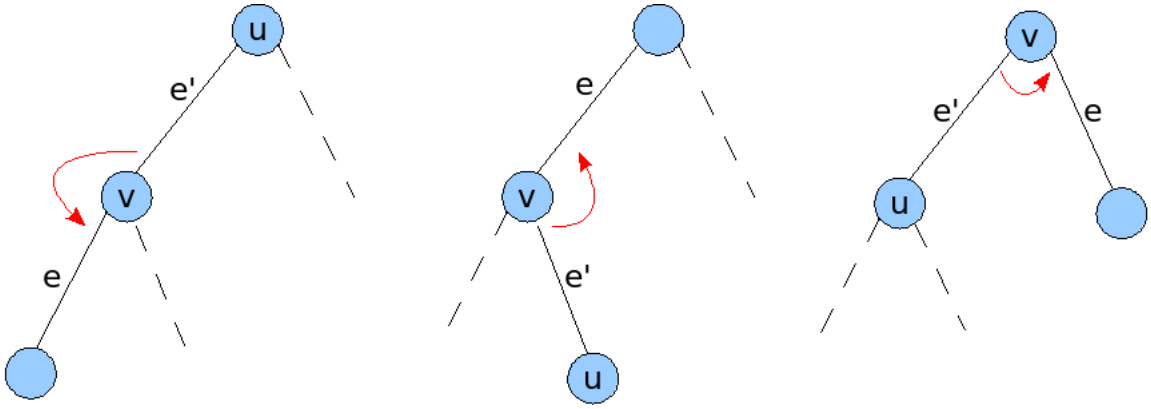
4.4 3. Phase

Die 3. Phase ist die komplizierteste und liefert endgültig einen geeigneten Separator. Wir werden in den nächsten Abschnitten auf die Teilschritte eingehen. Hierfür müssen die Graphknoten oberhalb von S_m und S_m selbst zu einem Knoten verschmolzen werden. Die kombinatorische Einbettung, insbesondere die Reihenfolge der Kanten, die mit dem neuen Wurzelknoten verbunden werden, muss erhalten bleiben. Desweiteren müssen die Knoten unterhalb von S_M und S_M gelöscht werden. Danach wird der neue Graph trianguliert, damit ein Vergrößern bzw. Verkleinern des Fundamentalkreises möglich ist. Dazu wird durch eine beliebige Nichtbaumkante ein Fundamentalkreis gebildet. Dieser Kreis wird entsprechend verkleinert bzw. vergrößert, bis die Anzahl der Knoten im Kreis mehr als $\frac{1}{3} \cdot n$ und weniger als $\frac{2}{3} \cdot n$ beträgt, wie im Abschnitt 3.4 beschrieben. In den weiteren Ausfhrungen möchten wir auf die einzelnen Schritte verdeutlichen.

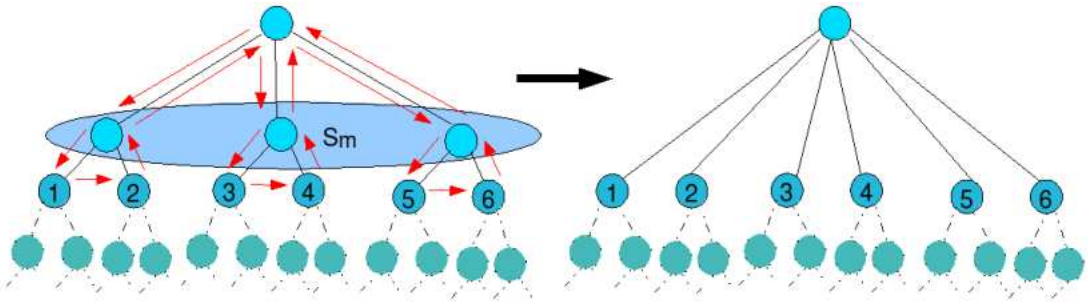
Das **Verschmelzen** der Knoten wird dadurch realisiert, dass alle Knoten oberhalb von S_m und S_m bis auf den Wurzelknoten, einschließlich der Kanten, die inzident zu den gelöschten Knoten sind, gelöscht werden. Nun müssen die Kanten, die zwischen dem alten S_m und S_{m+1} verlaufen, in der richtigen Reihenfolge (Gegenuhrzeigersinn) an die neue Wurzel und an die jeweiligen Knoten aus S_{m+1} angehängt werden. Um die Reihenfolge der Kanten auszuwählen, benutzen wir eine Traversierung[10] im ursprünglichen Graphen, die sich die Knoten aus S_{m+1} merkt und auch keine Knoten in tieferen Level besucht. Durch die kombinatorische Einbettung ist es uns möglich, von einem Knoten aus zu seiner inzidenten Kante, die Nachfolgerkante im Gegenuhrzeigersinn zu bestimmen. Dadurch können wir von einem Knoten u und seiner inzidenten Kante e' zur Kante e kommen, indem wir von u aus den anderen Knoten der Kante e' abfragen und die Nachfolgerkante von e' bzgl. v bestimmen, vergleiche hierzu die Skizzen auf der nächsten Seite. Dieser Schritt wird in der Traversierung dazu verwendet,

[10] Traversierung ist ein Verfahren, das eine Route in einen Baum durchläuft, um Knoten in einer bestimmten Reihenfolge auszugeben.

um den Baum abwärts zu gehen (linke Skizze), um den Baum aufwärts zu gehen (mittlere Skizze) und zur rechten Kante vom Knoten des gleichen Levels zu gehen (rechte Skizze). Der Schritt wird solange angewendet, bis sich der andere inzidente Knoten der Kante nicht in S_{m+1} befindet, beispielsweise falls wir uns beim Knoten u und seiner inzidenten Kante e' befinden, wird der Schritt nur angewendet, falls sich der Knoten v nicht in S_{m+1} befindet.



Wie schon angedeutet, wird der Baum durchlaufen bis zu einer Kante bzgl. eines Knotens in S_m. Dann wird geprüft, ob der andere inzidente Knoten dieser Kante in S_{m+1} liegt. Falls dies der Fall ist, wird der Knoten der Liste hinzugefügt, die später alle Knoten aus S_{m+1} in der richtigen Reihenfolge enthält. Anschließend wird die Nachfolgerkante gewählt. Falls der Knoten nicht in S_{m+1} liegt, wird mit dem obigen Schritt der Traversierung fortgefahren. Die untere Skizze zeigt, wie der aufspannende Baum des Graphen durchlaufen wird. Die roten Pfeile deuten den Weg der Traversierung an und die Nummern in den Knoten weisen auf die Reihenfolge hin, in welcher sie sich in der Liste befinden.

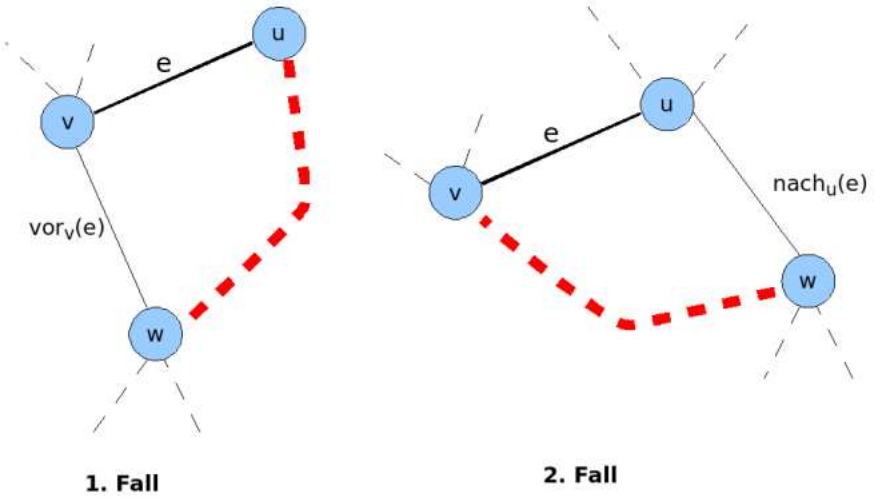


Nachdem wir die Liste mit der richtigen Reihenfolge der Knoten erstellt haben, können wir die Kanten einfügen und erhalten eine korrekte kombinatorische Einbettung des neuen Graphen. Desweiteren werden die Knoten bzw. inzidente Kanten zu diesen Knoten aus S_M und unterhalb von S_M gelöscht. Darauf werden wir nicht weiter eingehen, da es relativ intuitiv ist.

Die **Triangulierung** findet im kopierten, veränderten Graphen statt. Hierbei orientieren wir uns am technischen Bericht [Paj07]. Die Visualisierung eines triangulierten Graphen wird uns erschwert, da man nicht mehr allein mit geradlinigen Kanten auskommt. Dies werden wir jedoch später im Abschnitt 4.5 behandeln. Bei triangulierten Graphen besteht jede Facette aus einem Dreieck, da der triangulierte Graph ein kantenmaximaler, planarer Graph ist. Dies kann man sich veranschaulichen, wenn man sich vorstellt, dass eine Facette kein Dreieck ist. Daraus folgt, dass die Facette doch ein n-Eck ist und $n > 3$. Das kann aufgrund der Kantenmaximalität nicht sein, da man noch ein oder mehrere Kanten einfügen kann. Aus diesen Überlegungen läßt sich ein Algorithmus zur Triangulierung herleiten. Dazu ergänzt dieser bei jeder Kante die rechte Facette zu einem Dreieck, wobei darauf geachtet werden muss, dass die Planaritätseigenschaft erhalten bleibt, d. h. sich die Kanten nicht kreuzen. Hierfür werden bei jedem Knoten alle noch nicht markierten, inzidenten Kanten im Uhrzeigersinn durchgegangen. Beim Ergänzen der rechten Facette unterscheidet man drei Fälle, auf die wir kurz eingehen wollen.

1. **Fall:** Falls u als einzige inzidente Kante e, also keine weiteren inzidenten Kanten hat, fügt man die rot gestrichelte Kante zwischen u und w ein. Da es zwischen e und $vor_v(e)$ (Vorgängerkante von e bzgl. v) keine Kante geben kann, die die hinzugefügte Kante kreuzt.

2. **Fall:** Falls u mehrere inzidente Kanten außer e hat und keine Kante zwischen v und w existiert, so fügt man die rot gestrichelte Kante hinzu. Analog kann auch hier keine Kante zwischen e und $nach_u(e)$ (Nachfolgerkante von e bzgl. u) vorhanden sein. Ob eine Kante zwischen u und w existiert, wird geprüft, indem man alle Nachbarknoten[11] von v mit einem Index markiert. Falls dieser Index von w ungleich dem von v ist, kann es folglich keine Kante zwischen u und w geben.

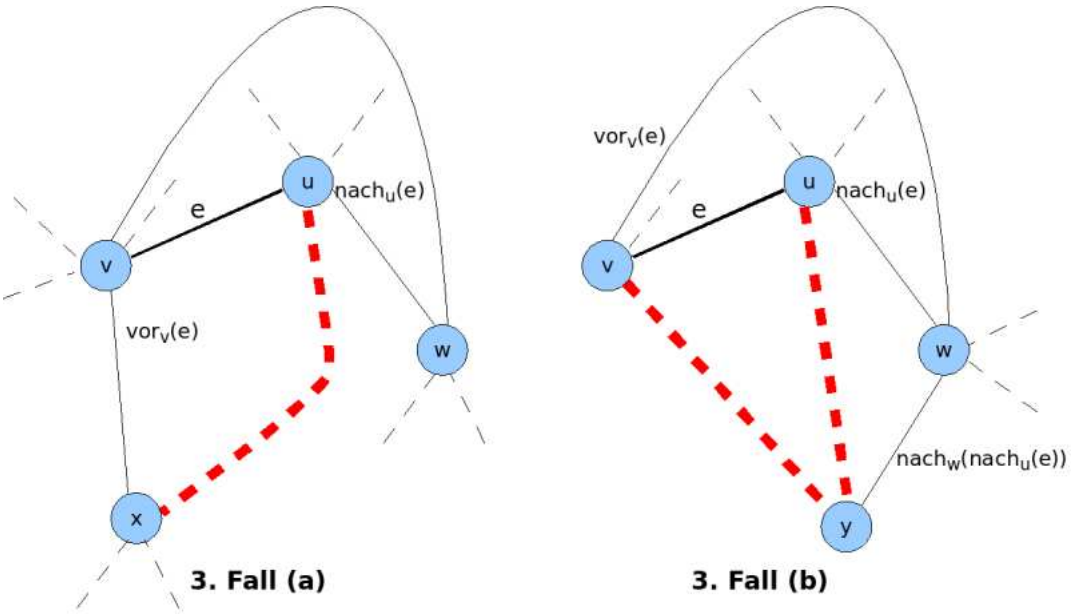


[11] Nachbarknoten sind von dem Knoten aus durch *nur eine* Kante erreichbar.

3. Fall: Der dritte Fall liegt vor, falls u mehrere inzidente Kanten besitzt und eine Kante zwischen den Knoten v und w existiert. Er unterteilt sich in zwei Teilfälle, Fall a, in dem nur eine Kante fehlt und b, in dem sogar zwei Kanten fehlen, damit die rechte Facette ein Dreieck bildet.

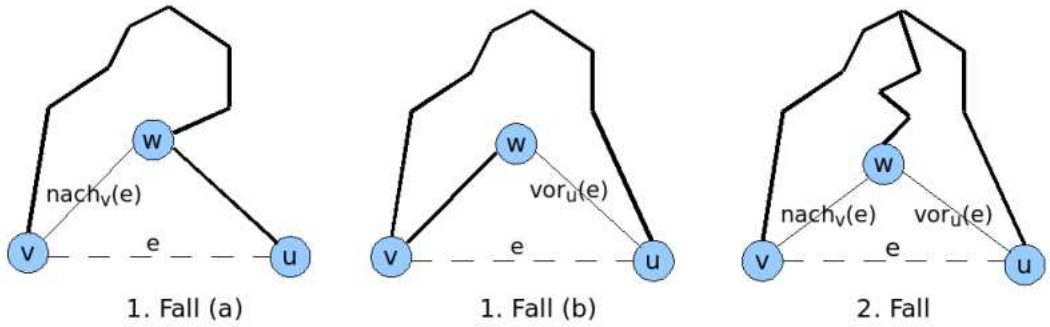
Fall a: Falls die Kante zwischen den Knoten v und w nicht $vor_v(e)$ liegt, dann wird eine Kante zwischen den Knoten u und x eingefügt. Da es auch hier wieder keine Kante geben kann, die die neu eingefügte Kante kreuzt.

Fall b: Falls die Kante zwischen den Knoten v und w $vor_v(e)$ sich befindet, dann muss eine Kante zwischen den Knoten u und y, aber auch eine weitere Kante zwischen den Knoten v und y einfügt werden. Hier ist eine Trennung des Falles in a und b notwendig, da sonst der Knoten w dem Knoten x in Fall a entsprechen würde und wir dann eine weitere Kante zwischen den Knoten u und w hinzufügen würden.



Das Verkleinern und Vergrößern des Kreises beginnt damit, dass wir eine beliebige Nichtbaumkante wählen, um mit dieser einen Fundamentalkreis zu bilden. Danach stellen wir fest, ob wir den Kreis verkleinern oder vergrößern müssen. Dies erfordert eine Zählung der Knoten, die sich innerhalb bzw. außerhalb des Kreises befinden. Die Zählung der Knoten werden wir im nächsten Abschnitt erläutern. Wir haben nicht festgestellt haben, welche Knotenmenge im Inneren und welche im Äußeren des Kreises liegt. Wir definieren deshalb die Mengen einfach mit Linkes bzw. Rechtes des Kreises, was von der Richtung der Knotenzählung abhängt. Die Unterscheidung zwischen Innerem und Äußerem des Kreises ist nicht notwendig und auch schwer zu realisieren, da wir mit der kombinatorischen Einbettung arbeiten und in dieser keine Koordinaten der Knoten gegeben sind. Die Orientierung ist eindeutig, wir legen sie fest von einem Knoten und einer dazu inzidenten Nichtbaumkante, die uns die Seiten Rechtes (in Richtung der Vorgängerkante) und Linkes (in Richtung der

Nachfolgerkante) eindeutig zuordnen lässt. Falls wir z. B. das Linke verkleinern wollen, wissen wir, dass wir in der linken Facette eine Nichtbaumkante wählen müssen. Genauer gesagt, unterteilt sich das in zwei Fälle, wobei wir annehmen, dass wir das Linke verkleinern wollen. Die anderen funktioneren entsprechend, gegebenenfalls mit der linken Facette. Die zwei Fälle unterscheiden, ob die anderen (nicht die gerade ausgewählte Kante, die einen Kreis bildet) Kanten der Facette (Dreieck) eine Baumkante enthalten. In den unteren Skizzen ist e die gewählte Nichtbaumkante vom Knoten v aus.



1. Fall: Falls eine der anderen Kante der Facette eine Nichtbaumkante darstellt, wird als nächstes die Kante gewählt, die ein Nichtbaumkante ist, d. h. wie im 1. Fall (a) $nach_v(e)$ oder wie im 1. Fall (b) $vor_u(e)$. Aus den Skizzen wird deutlich, dass im 1. Fall (a) das Innere bzw. Linke gleich bleibt und das Äußere bzw. Rechte um einen Knoten (nämlich u) zunimmt, dies wäre bei $vor_u(e)$ ebenso möglich. Bei 1. Fall (b) wird das Innere bzw. das Linke um einem Knoten (den Knoten w) kleiner und das Aüßere bleibt gleichgroß.

2. Fall: Falls beide anderen Kanten der Facette Nichtbaumkanten sind, wählen wir die Nichtbaumkante, die mehr Knoten im Linken beinhaltet. Der Grund liegt darin, dass wir beide Mengen Linkes und Rechtes ausbalancieren wollen, d. h. sie haben jeweils zwischen $\frac{n}{3}$ und $2 \cdot \frac{n}{3}$ Knoten. Aus der Skizze wird klar, das Linke hat sich stark verkleinert.

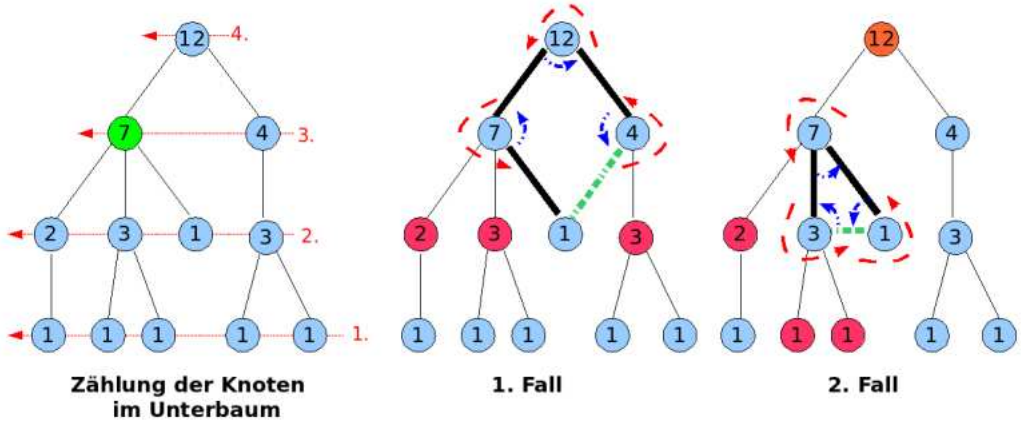
Desweiteren kann es vorkommen, dass unsere Seiten wechseln, d. h. bei einer Wahl der Nichtbaumkante zur nächsten, das Linke dann zum Rechten wird. Dies verursacht aber keine Probleme, solange dies nicht innerhalb von einem Schritt passiert. Deswegen haben wir uns weitere Fallunterscheidungen gespart.

Das Zählen der Knoten beginnt damit, dass wir bei jedem Knoten des Baumes speichern, wieviele Knoten sich in seinem Unterbaum[12] befinden. Dies realisieren wir dadurch, in dem wir die gespeicherte Reihenfolge der besuchten Knoten von der Breitensuche rückwärts benutzen. Da wir bereits wissen, dass alle Knoten, die sich im Unterbaum befinden schon bearbeitet wurden und die gespeicherte Anzahl der Knoten im Unterbaum von den Kindknoten[13] aufsummiert werden können. Dabei ist noch zu beachten, dass wir den Knoten selbst

[12] Ein Unterbaum ist ein Teilgraph eines Baumes, der selbst als kompletter Baum angesehen werden kann.

[13] Ein Kindknoten ist verbunden mit dem Knoten, der ein Level höher liegt.

noch nicht mitgezählt haben. Deshalb müssen wir diesen zusätzlich addieren. Im Beispiel (unteres Bild) ist die Reihenfolge der Knoten mit rot gestrichelten Pfeilen und roten Nummer gekennzeichnet. Desweiteren ist in jedem Knoten die Anzahl der Knoten im Unterbaum dargestellt. Zum Beispiel wird bei dem grünen Knoten die Anzahl der Knoten im Unterbaum von den Kindknoten aufaddiert und zusätzlich den Knoten selbst, also $1 + 3 + 2 + 1 = 7$.



Bei der eigentlichen Zählvorgang müssen wir zwei Fälle unterscheiden, ob die Wurzel auf dem Kreis liegt oder nicht. Die Richtung, in der die Zählung durchgeführt wird, entscheidet darüber, was Linkes und was Rechtes des Kreises wird. Dazu benutzen wir die kombinatorische Einbettung, um zu erkennen, auf welcher Seite (Linkes oder Rechtes) des Kreises die Knoten liegen.

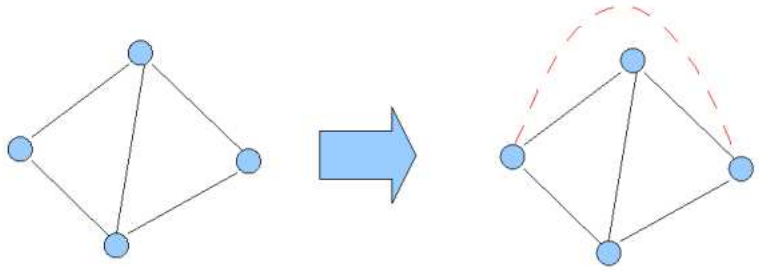
1. **Fall:** Falls die Wurzel auf dem Kreis liegt, beginnen wir mit der Nichtbaumkante und bestimmen die Nachfolgerkante. Wir addieren so lange die Anzahl der Knoten im Unterbaum von dem anderen Knoten dieser inzidenten Kante, der nicht auf dem Kreis liegt, bis wir wieder eine Kreiskante erreichen. Diese Kreiskante müssen wir uns merken, damit wir von dieser analog das Linke des Kreises bestimmen. Danach können wir mit dieser Kreiskante und dem anderen inzidenten Knoten dieser Kante sukzessiv nach dem gleichen Prinzip fortfahren, bis wir am Ende die Nichtbaumkante erreichen. Im Beispiel (oberes mittleres Bild) besteht das Rechte aus dem Äußeren des Kreises und das Linke beinhaltet keine Knoten und stellt das Innere des Kreises dar, wenn wir bei dem Knoten anfangen, der 4 Knoten in seinem Unterbaum hat. Um die Anzahl der Knoten im Rechten des Kreises zu erhalten, addieren wir die Anzahl der Knoten im Unterbaum der roten Knoten, also ergibt es $3 + 2 + 3 = 8$ Knoten im Rechten bzw. Äußeren des Kreises.

2. **Fall:** Falls die Wurzel nicht auf dem Kreis liegt, läuft der Prozess fast analog wie im ersten Fall, wobei wir darauf achten müssen, dass der Knoten, von dem wir die Anzahl der Knoten im Unterbaum addieren, ein Kindknoten des Kreisknotens sein muss. Ansonsten müssen wir die Anzahl der Knoten im Unterbaum vom Kreisknoten von denen des anderen Knotens subtrahieren. Im Beispiel (oberes rechtes Bild) addieren wir analog zum ersten Fall, die Anzahl der Knoten im Unterbaum der roten Knoten. Bei

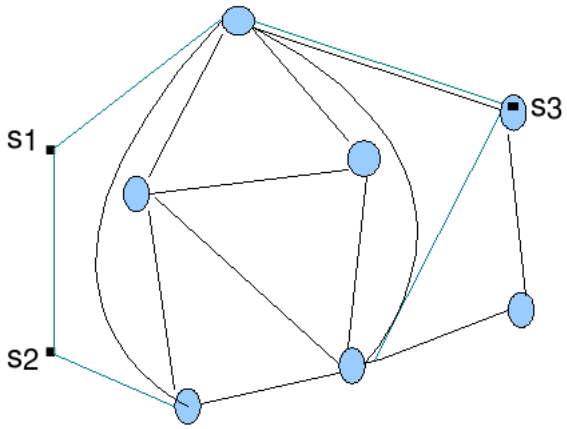
dem orangenen Wurzelknoten müssen wir die Anzahl der Knoten im Unterbaum des entsprechenden Kreisknotens von der Anzahl der Knoten im Unterbaum des Wurzelknotens subtrahieren, also $12 - 7 = 5$. Summiert ergibt dies $5 + 2 + 1 + 1 = 9$ Knoten im Äußeren des Kreises.

4.5 Visualisierung

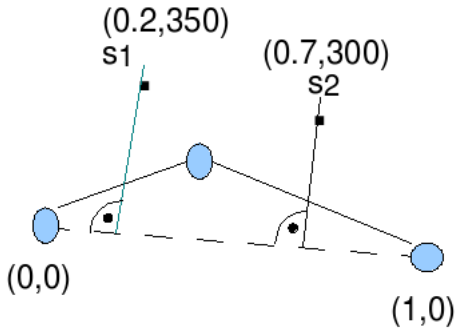
Die Visualisierung erweist sich als Hindernis, da sich zwar der Graph relativ einfach in der geometrischen Einbettung realisieren lässt (da die Koordinaten vorliegen), jedoch bei der kombinatorischen Einbettung mit einem triangulierten Graphen, wie schon angedeutet, nicht mehr nur mit geradlinigen Kanten umgesetzt werden kann. Dies wird in der unteren Skizze verdeutlicht. Da die aüßere (unbegrenzte) Facette noch kein Dreieck bildet, muß man eine Kante hinzufügen, die aber nur eine Kurve sein kann, da sie sonst eine oder mehrere andere Kanten schneidet. Man könnte z. B. die rot gestrichelte Kante hinzufügen.



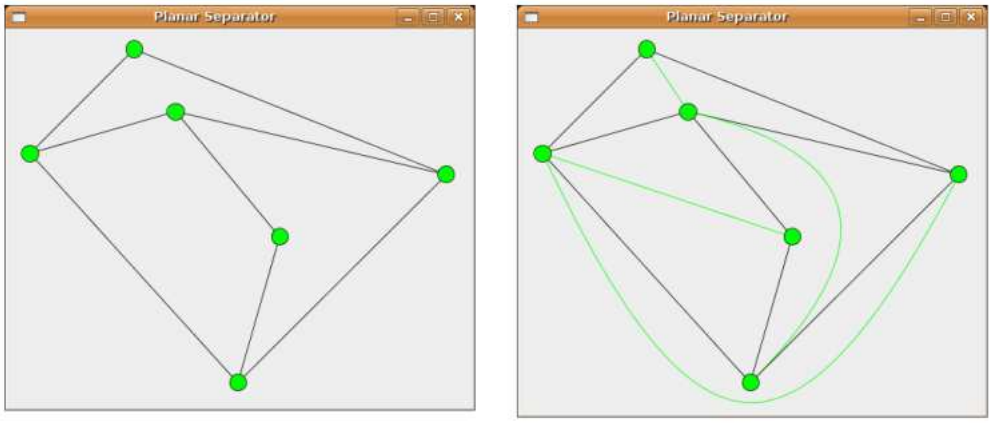
Deshalb benutzen wir Kurven als Kanten. Hier haben wir Bézierkurven verwendet. Das sind Kurven, die anhand von Anfangs-, End- und Stützpunkt(en) gezeichnet werden können. Vorwiegend verwenden wir quadratische Bézierkurven mit einem Stützpunkt für Kanten, die nicht an der äußeren (unbegrenzten) Facette liegen und kubische Bézierkurven mit zwei Stützpunkten für die anderen Kanten. Im Falle, dass keine Kurve erforderlich sind, verwenden wir natürlich weiterhin geradlinige Kanten.



Desweiteren sind die Koordinaten, die JUNG verwendet hat, relativ zum Kantenanfang und Kantenende, d. h. der Anfangspunkt einer Kante wird mit $(0, 0)$ und der Endpunkt mit $(1, 0)$ bezeichnet. Die zweite Koordinate kann man anhand einer orthogonalen Projektion bestimmen, sie ist in Pixel angegeben, also nicht relativ zur Kantenlänge wie die erste Koordinate. Um die orthogonale Projektion zu berechnen, verwenden wir unser Vorwissen aus der Vorlesung Lineare Algebra, man kann dies auch in unserem LA-Skript[DW04] nachlesen.

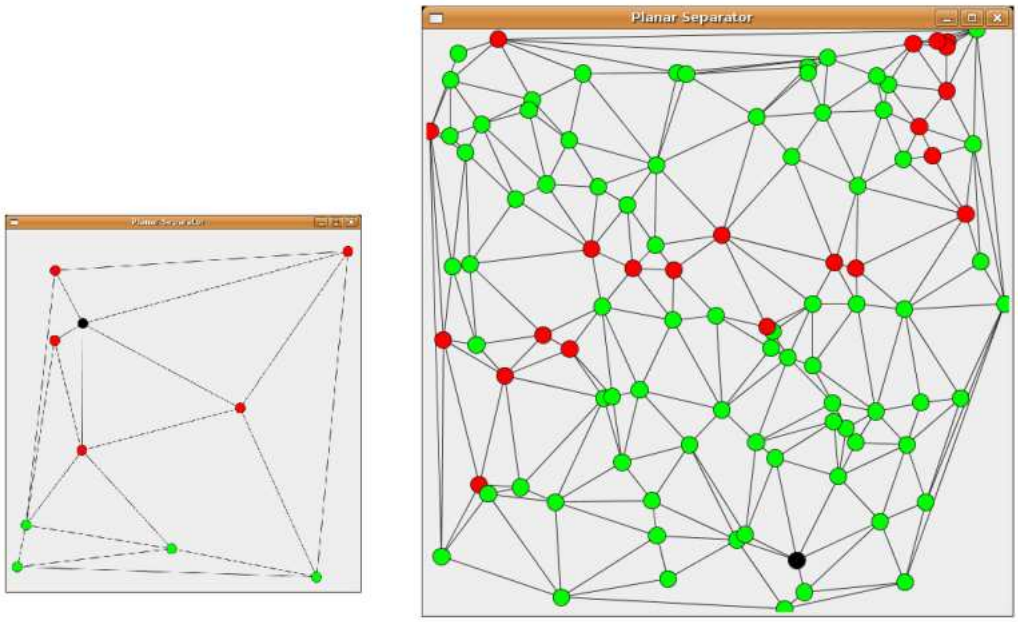


Leider ist es uns trotz der Anstrengungen nicht gelungen, einige Spezialfälle zu lösen, z. B. falls der Knoten um den die Kurve (Kante) gezeichnet werden soll, außerhalb von $[0, 1]$ bei der ersten Koordinate liegt. Dies hat JUNG leider nicht unterstützt. Desweiteren bereitete es uns Schwierigkeiten, eine Kurve um eine weiter Kurve zu zeichnen, da die Parameter (Koordinaten der Stützpunkte) nicht leicht zu bestimmen sind. Die restlichen Parameter bei den Kurven um geradlinige Kanten konnte wir auf Grund von Erfahrungswerten, der Länge der Kante und dem Winkel am Anfangspunkt der Kante zwischen Stützpunkt und Endpunkt der Kante berechnen. Das untere Bild zeigt dies an einem praxisnahen Beispiel: Auf dem linken, der noch nicht triangulierte Graph und auf dem rechten, der triangulierte Graph mit den grünen hinzugefügten Kanten.



5 Experimente

Wir haben bei den Experimenten hauptsächlich Delaunay Graphen auf die Größe des Seperators getestet. Das untere Bild zeigt zwei Delaunay Graphen, der Linke mit 10 und der Rechte mit 100 Knoten. Die schwarzen Knoten entsprechen den ausgewählten Wurzelknoten und die roten Knoten stammen aus dem μ-Level, das bereits einen geeigneten Separator bildet.

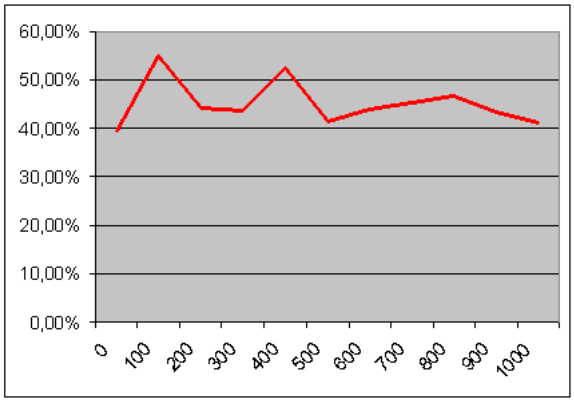


5.1 Hilfsmittel

Im Wesentlichen verwenden wir für die Experimente zwei Hilfsmittel, yEd[yEd] und Graphgen. yEd diente uns als Grapheneditor, mit dessen Hilfe wir zahlreiche Graphen erstellt haben, um Problemfälle z. B. der Visualisierung zu testen. Graphgen bietet als Graphengenerator die Möglichkeit, zahlreiche verschiedene Graphenarten zu generieren, wie Delaunay Graphen, Dreieckgraphen, Secksgitter und viele weitere. Für die Experimente sind die Delaunay Graphen von der Struktur am interesantesten.

5.2 Ergebnisse

Das Diagramm auf der nächsten Seite zeigt, wie sich mit steigender Knotenanzahl (horizontale Achse) die Separatorgröße (vertikale Achse) entwickelt. Der prozentuale Anteil der maximal möglichen Separatorgröße ($4 \cdot \sqrt{n}$ Knoten) ist auf der vertikalen Achse aufgetragen. Der Verlauf des Diagramms macht deutlich, dass sich die Separatorgröße prozentual exakt zwischen 40% und 55% bewegt. Hierbei wurden Delaunay Graphen getestet.



6 Ausblick

Die Visualisierung des triangulierten Graphen ist eindeutig verbesserungsfähig, wobei das keineswegs leicht zu realisieren ist. Zusätzlich hätten wir noch viele weitere interessante Experimente durchführen können. Wobei wir verschiedene weitere Größen hätten testen können, wie:

- Die Balance der Mengen V_1 und V_2
- Welche Phase einen Separator liefert
- Wieviele Schritte beim Vergrößern bzw. Verkleinern der Fundamentalkreise benötigt werden

Weiterhin wären auch noch die Fragestellungen interessant:

- Wie sich die Wahl des Wurzelknotens auf die Phase, in der ein Separator geliefert wird, auswirkt?
- Welche Nichtbaumkante am besten gewählt werden sollte, um möglichst wenige Schritte beim Vergrößern bzw. Verkleinern der Fundamentalkreise zu benötigen?

Nicht zuletzt wäre es aufschlußreich, ob es eventuell eine gute Heuristik für allgemeine planare Graphen geben kann.

Literatur

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Breitensuche*, chapter 22.2, pages 535–543. MIT Press and McGraw-Hill, second edition, 2001.

[DW04] V. Drumm and W. Weil. *Orthonormalbasen und Orthogonalprojektion*, chapter 5.2, pages 235–246. Mathemathisches Institut II - Universität Karlsruhe, Oktober 2004.

[jav] Java 1.5. `http://java.sun.com/`.

[juna] Jung java universal network/graph framework. `http://jung.sourceforge.net`.

[junb] Jungx graphenbibliotek. `http://i11www.iti.uni-karlsruhe.de/teaching/WS_0607/planalgo/jungX/doc/index.html`.

[LT77] Richard J. Lipton and Robert E. Tarjan. *A separator theorem for planar graphs*. Forschungsbericht. Stanford, CA, USA : Stanford University 1977.

[Paj07] Thomas Pajor. *Beschreibung eines Algorithmus zur Triangulierung eines planaren Graphen*. Technischer Bericht. Februar 2007.

[Wag06] Dorothea Wagner. Vorlesungsskript algorithmen für planare graphen. `http://i11www.iti.uni-karlsruhe.de/teaching/SS_06/planalgo/skript/algo-plan.pdf`, SS 2006.

[yEd] yed. `http://www.yworks.com/de/products_yed_about.htm`.