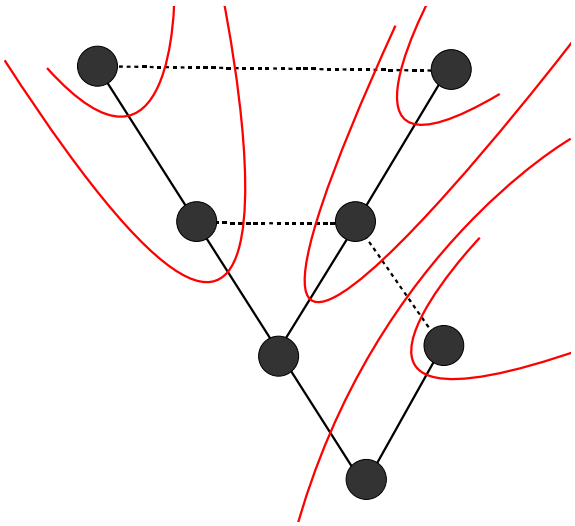
Minimale Schnitte und Schnittbäume

Studienarbeit von Myriam Freidinger



Betreuer: Prof. Dr. Dorothea Wagner, Robert Görke

ITI Prof. Dr. Dorothea Wagner, Universität Karlsruhe

23. Februar 2007

Danksagung

Ich möchte mich an dieser Stelle bei Prof. Dr. Dorothea Wagner und Robert Görke für die Unterstützung bei der Erstellung dieser Arbeit bedanken. Ihrer großen Hilfe bei Themenwahl und Literatursuche ist es zu verdanken, dass ich meinen Einstieg in das selbstständige wissenschaftliche Arbeiten als sehr angenehm empfunden habe.

Ich versichere hiermit wahrheitsgemäß, die Arbeit bis auf die dem Aufgabensteller bereits bekannte Hilfe selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

Myriam Freidinger
Karlsruhe, 23. Februar 2007

Zusammenfassung

Ziel dieser Arbeit ist es zu untersuchen, in wie weit sich der Algorithmus von Stoer und Wagner dazu verwenden lässt, einen minimalen Schnittbaum zu erzeugen.

Der Algorithmus von Stoer und Wagner berechnet den minimalen Schnitt in einem Graphen und erzeugt im Laufe dessen insgesamt $n - 1$ unterschiedliche Schnitte des Graphen. Eine genauere Untersuchung dieser Schnitte ergab leider, dass sie sich nicht dazu eignen, einen minimalen Schnittbaum zu erstellen. Sie sind zwar linear unabhängig und erfüllen die notwendige Eigenschaft, dass sie sich paarweise nicht kreuzen, das Gewicht des aus ihnen berechneten Schnittbaumes ist aber nicht minimal. Auch als Heuristik erzielt der Algorithmus keine guten Resultate.

Als zweiten Teil der Arbeit habe ich Heuristiken zu Schnittbäumen untersucht. Die Heuristiken des maximalen kurzen Baumes und des daraus resultierenden optimierten Baumes laufen schnell und zuverlässig auf den von mir getesteten Graphen. Eine Gütegarantie mit Faktor 2 sichert konstant gute Ergebnisse.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Minimale Schnitte sind wichtig um die Verlässlichkeit eines Netzwerkes zu bestimmen und werden deshalb schon seit langem ausgiebig untersucht. Sie sagen viel über den Zusammenhang eines Graphen aus und darüber, wie stark die einzelnen Ecken untereinander verbunden sind. Minimale Schnittbäume speichern auf sehr kompakte Art und Weise alle minimalen Schnitt in einem Graphen.

Schnittbaumalgorithmen werden zum Beispiel zur Einteilung eines Graphen in Cluster [FlTaTs03] verwendet. Es existieren auch Implementierungen von TSP-Algorithmen, die minimale Schnittbäume benutzen [GoTs01].

Meine Arbeit gliedert sich in fünf Abschnitte. Im ersten Abschnitt werde ich einige grundlegende Begriffe und Eigenschaften von Graphen und Schnitten einführen. Die Theorie der Schnittbäume und ihre historische Entwicklung folgen im zweiten Abschnitt. Anschließend untersuche ich, in wie weit sich Min-Schnitt-Algorithmen dazu verwenden lassen eine möglichst leichte Schnittbasis zu erzeugen. Dabei betrachte ich fast ausschließlich den Algorithmus von Stoer und Wagner. Im vierten Abschnitt werde ich dann Heuristiken zur Berechnung von leichten Schnittbasen vorstellen die ich im fünften Abschnitt an einigen Graphen teste.

1.1 Grundlegende Begriffe und Eigenschaften

Dieser Abschnitt dient dazu, die benötigten Begriffe rund um Graphen und Schnitte einzuführen.

Definition 1.1 (Graph)**:**
Ich betrachte immer einfache, ungerichtete *Graphen* $G = (V, E)$ mit Gewichtsfunktion $c : E \to \mathbb{R}^+$. Die Graphen dürfen folglich weder Schleifen noch Mehrfachkanten enthalten. Der Parameter V bezeichnet die Menge der Ecken, E die Menge der Kanten, $n = |V|$ die Anzahl der Ecken und $m = |E|$ die Anzahl der Kanten. Eine Kante $e \in E$ wird häufig als Tupel $e = \{v, w\}$ dargestellt, wobei $v, w \in V$ die inzidenten Ecken der Kante sind. Die Funktion $c : E \to \mathbb{R}^+$ legt die Gewichte der Kanten fest. Falls c nicht explizit angegeben ist, so haben alle Kanten ein einheitliches Gewicht von 1.

Die betrachteten Graphen seinen außerdem o.B.d.A zusammenhängend, da andernfalls alle betrachteten Probleme auf den einzelnen Zusammenhangskomponenten gelöst werden können. Die Gesamtlösungen setzen sich dann jeweils aus den Einzellösungen zusammen.

Definition 1.2 (Schnitt)**:**
Ein *Schnitt* S in einem Graphen $G = (V, E)$ ist eine Partition des Graphen in S und $V \backslash S$. Das *Gewicht* eines Schnittes S in einem Graphen mit Gewichtsfunktion c ist die Summe der Gewichte aller Kanten, die eine Endecke in S und eine in $V \backslash S$ haben.

$$c(S) = \sum_{\substack{\{v,w\} \in E: \\ v \in S \wedge w \in V \backslash S}} c(\{v, w\})$$

Zwei Ecken v, w werden von einem Schnitt S *getrennt*, wenn $v \in S$ und $w \in V \backslash S$ oder umgekehrt. Eine Kante $\{v, w\}$ zwischen zwei Ecken, die vom Schnitt getrennt werden, *kreuzt* den Schnitt.

Definition 1.3:
Zwei Schnitte $(S, V \backslash S)$ und $(T, V \backslash T)$ *kreuzen* sich, falls keine der folgenden Mengen leer ist: $S \cap T$, $S \cap V \backslash T$, $V \backslash S \cap T$ und $V \backslash S \cap V \backslash T$

Zeichnet man die Schnitte wie in Abbildung 1.1, so schneiden sich zwei Schnitte genau dann nicht, wenn man sie ohne Kreuzung der zugehörigen Linien zeichnen kann. In der Abbildung kreuzen sich also keine der gezeichneten Schnitte.

Definition 1.4 (Darstellung eines Schnittes als Vektor über $\mathbb{F}_2$)**:**
Die Menge aller Teilmengen von E kann als m-dimensionaler Vektorraum über $\mathbb{F}_2$ aufgefasst werden. Dazu wird eine feste Reihenfolge der Kanten

$$E = \{e_1, e_2, e_3, \ldots e_m\}$$

gewählt. Eine Menge von Kanten $X \subseteq E$ entspricht anschließend dem Vektor

$$\vec{x} = (x_i)_{i=1,\ldots,m} \text{ mit } \quad x_i = \begin{cases} 1 & \text{falls } \quad e_i \in X \\ 0 & \text{falls } \quad e_i \notin X \end{cases}$$

Mit Hilfe dieser Darstellung können Teilmengen von E addiert werden.

Ein Schnitt kann auch eindeutig durch die Kanten beschrieben werden, die ihn kreuzen. In dieser Form können zwei Schnitte, genau wie alle anderen Teilmengen von E addiert werden. Zwei Beispiele hierfür sind in Abbildung 1.1 dargestellt. Anschaulich entspricht die Addition zweier Schnitte der Vereinigung ihrer Kanten ohne die Menge der Kanten, die in beiden Schnitten enthalten sind (symmetrische Differenz).

Werden die Schnitte wieder als Partition definiert, so ergibt sich

$$S + T = (S \cap T) \cup (V \backslash S \cap V \backslash T)$$

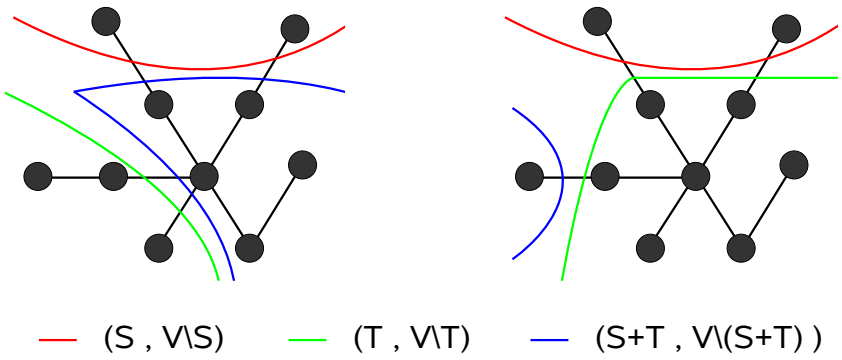


Abbildung 1.1: Zwei Beispiele für die Addition von Schnitten

Lemma 1.1. *Die Menge der Schnitte in einem Graphen bildet einen Untervektorraum des $\mathbb{F}_2-$Vektorraumes der Teilmengen von E.*

Beweis. In jedem Graphen gibt es den trivialen Schnitt $(V, \emptyset)$. Die Menge der Schnitte ist also nicht leer. Es kann gezeigt werden, dass die Summe zweier Schnitte stets wieder einen Schnitt ergibt (siehe [Gib85]). Die Skalarmultiplikation mit 0 ergibt immer den trivialen Schnitt $(V, \emptyset)$, die Multiplikation mit 1 den Schnitt selbst. Außerdem ist jeder Schnitt zu sich selbst invers. q.e.d.

Definition 1.5 (linear unabhängig)**:**
Analog zur Linearen Algebra ist eine Menge von Schnitten *linear unabhängig*, falls es keine nicht triviale Linearkombination des trivialen Schnittes aus ihnen gibt bzw. falls sich keiner der Schnitte durch einen Linearkombination der anderen erzeugen lässt. Andernfalls heißen die Schnitte linear abhängig.

Da die Schnitte als Vektorraum über $\mathbb{F}_2$ betrachtet werden, sind folgende Aussagen äquivalent:

- Die Schnitte in $M := \{S_1, S_2, \ldots, S_n\}$ sind linear abhängig.
- Es gibt eine Teilmenge $\{S_{i_1}, S_{i_2}, \ldots, S_{i_l}\} \subseteq M$ in der jede Kante ein gerade Anzahl mal vorkommt.

Definition 1.6 (Schnittbasis)**:**
Eine *Schnittbasis* B ist eine linear unabhängige Menge von Schnitten, aus der sich jeder Schnitt des Graphen erzeugen lässt. Ein Schnitt kann dann eindeutig als Linearkombination $S_{i_1} + S_{i_2} + \cdots + S_{i_l}$ mit $S_{i_j} \in B$ geschrieben werden.

Lemma 1.2. *Eine Schnittbasis enthält immer $n-1$ verschiedene Schnitte.*

Definition 1.7 (Gewicht einer Schnittbasis)**:**
Das Gewicht eines Schnittes S, aufgefasst als Kantenmenge, ist nach Definition 1.2 gegeben durch

$$c(S) := \sum_{e \in S} c(e)$$

Das Gewicht einer Schnittbasis $B = \{S_1, \ldots, S_{n-1}\}$ ist die Summe der Gewichte der Elemente der Basis.

$$c(B) = \sum_{i=1}^{n-1} c(S_i) = \sum_{i=1}^{n-1} \sum_{e \in S_i} c(e)$$

Definition 1.8 (minimaler Schnitt, der u und v trennt)**:**
Ein Schnitt S heißt minimaler Schnitt, der u und v trennt, falls er u und v trennt und das kleinste Gewicht unter allen Schnitten mit dieser Eigenschaft hat.

Definition 1.9 (Eckenschnitt)**:**
Ein *Eckenschnitt* ist ein Schnitt $(v, V \backslash \{v\})$, also ein Schnitt bei dem eine Hälfte der Partition nur aus einer einzelnen Ecke besteht.

Kapitel 2

Schnittbäume

Die Definition des Schnittbaumes kommt ursprünglich aus dem Bereich des „Network Flows“. Ein Netzwerk besteht aus einem im Allgemeinen gerichteten Graphen $G = (V, E)$ mit Kantengewichten $c : E \to \mathbb{R}^+$ und zwei Ecken $s, t \in V$. In einem solchen Netzwerk sucht man nach einem maximalen Fluss, indem man, anschaulich gesprochen, Gewicht von s nach t über die Kanten des Graphen schiebt. Zu beachten ist dabei, dass man das Gewicht zwar auf mehrere Wege verteilen, aber unterwegs weder Gewicht verlieren noch dazugewinnen darf. Außerdem kann über jede Kannte e nur maximal ein Gewicht von $c(e)$ transportiert werden.

Dieses Problem entwickelte sich weiter zu dem Problem „Multi-Terminal Network Flow“, in welchem man für jedes Paar von Ecken u und v aus V einen solchen maximalen Fluss von u nach v sucht.

Ford und Fulkerson bewiesen 1956 in [FoFu56] das sogenannte „Max-Flow Min-Cut Theorem“, das besagt, dass der Wert eines maximalen Flusses von s nach t gleich dem minimalen Gewicht eines s-t-Schnittes in einem Netzwerk ist. Außerdem kann man aus einem maximalen Fluss den zugehörigen minimalen Schnitt gleichen Gewichts ablesen. Damit ist die Suche nach einer Lösung des „Multi-Terminal Network Flow“-Problems äquivalent zu der Suche nach einem minimalen Schnitt für jedes Paar von Ecken u und v.

Um eine solche Menge von Schnitten zu Berechnen sind im Prinzip $n \cdot (n-1)$ Flussberechnungen notwendig. Für den Fall eines ungerichteten Graphen (d.h. für ein Netzwerk mit $c((u, v)) = c((v, u))$) zeigten Gomory und Hu 1961 in [GH61], dass bereits $n-1$ minimale Schnitte in einem Graphen ausreichend sind, um für jedes Paar von Ecken einen Schnitt zu erhalten, der diese beiden Ecken minimal trennt. Ich werde in dieser Arbeit ebenfalls ausschließlich Schnitte in ungerichteten Graphen untersuchen und kann damit auf den Algorithmus von Gomory und Hu zur exakten Lösung des Problems zurückgreifen.

2.1 Der Gomory-Hu-Algorithmus

Das Besondere an den Schnitten, die vom Gomory-Hu-Algorithmus berechnet werden, ist, dass sie sich paarweise nicht kreuzen und daher auf sehr kompakte Art und Weise gespeichert werden können.

Definition 2.1 (Baum-Schnittbasis)**:**
Eine *Baum-Schnittbasis* ist eine Schnittbasis, deren Schnitte sich paarweise nicht kreuzen.

Definition 2.2 (Schnittbaum)**:**
Ein *Schnittbaum* oder auch *Gomory-Hu-Baum* eines Graphen $G = (V, E)$ ist eine kompakte Darstellung einer Baum-Schnittbasis und der Gewichte ihrer Schnitte. Er besteht aus einem aufspannenden Baum des vollständigen Graphen auf der Eckenmenge V. Das Entfernen einer beliebigen Baumkante lässt den Baum in zwei Zusammenhangskomponenten S und $V \backslash S$ zerfallen. Dadurch induziert jede Baumkante einen Schnitt $(S, V \backslash S)$ im Graphen. Das Gewicht dieses Schnittes wird als Gewicht der entsprechenden Baumkante gespeichert.

Zu beachten ist, dass die Kanten des Schnittbaumes keine Teilmenge der Kantenmenge des Graphen sein müssen. Für die Erstellung stehen alle Kanten des vollständigen Graphen K_n zur Verfügung. Das Problem der Suche nach einem *fundamentalen Schnittbaum*, also einem Schnittbaum, der nur Kanten verwenden darf, die in E enthalten sind, werde ich in dieser Arbeit nicht untersuchen.

Für das Verständnis des Zusammenhangs von minimalen Schnittbasen, Baumschnittbasen und Schnittbäumen sind die folgenden zwei Aussagen hilfreich, die Florentine Bunke in [Bun06] beweist:

Lemma 2.1. *Sei D eine Baum-Schnittbasis des Graphen G. Dann existiert ein aufspannender Baum des zugrundeliegenden vollständigen Graphen K_n, der diese Schnittbasis darstellt.*

Theorem 2.2. *Das Problem der minimalen Schnittbasis kann durch jeden Algorithmus gelöst werden, der einen minimalen Schnittbaum berechnet.*

Nach Lemma 2.1 kann eine Baum-Schnittbasis also immer durch einen Schnittbaum dargestellt werden. Der Algorithmus von Gomory und Hu berechnet nach Theorem 2.2 eine minimale Schnittbasis.

Für die Durchführung des Algorithmus bedarf es der Einführung einer neuen Graphoperation.

Definition 2.3 (Ecken verschmelzen)**:**
Sei $G = (V, E)$ ein ungerichteter Graph mit Gewichtsfunktion $c : E \to \mathbb{R}^+$, $X \subseteq V$ eine Menge von Ecken in V. Der Graph $G' := (V', E')$ entsteht durch verschmelzen der Ecken aus X zu einer neuen Ecke $\{X\}$,
d.h. $V' = V \backslash X \cup \{X\}$ und

$$E' = \{\, \{u, v\} \in E \mid u \notin X \wedge v \notin X \} \cup \{\, \{u, \{X\}\} \mid \exists v \in X \text{ mit } \{u, v\} \in E\}$$

Für Kanten aus E bleibt $c(\{u,v\})$ erhalten. Die neuen Kanten $\{u,\{X\}\}$ bekommen das Gewicht

$$c(\{u,\{X\}\}) = \sum_{v\in X,\{u,v\}\in E} c(\{u,v\})$$

Kanten innerhalb von X entfallen beim Verschmelzen komplett.

Algorithmus 1 : Gomory-Hu-Algorithmus (Graph $G=(V,E)$)

1 $T := (V_T, E_T)$ mit $V_T = \{V\}, E_T = \emptyset$ `// also` $|V_T| = 1$
2 **solange** $|V_T| \neq |V|$ **wiederhole**
3 Wähle $Q \in V_T$ mit $|Q| \geq 2$
4 Betrachte den Graphen G', der entsteht indem alle $Q' \in T, Q' \neq Q$ zu einer Ecke verschmolzen werden.
5 Wähle $x, y \in Q$.
6 $(A,B) :=$ minimaler x-y-Schnitt in G'.
7 $V_T := V_T \backslash \{Q\} \cup \{A \cap Q, B \cap Q\}$

$$\begin{aligned} E_T \quad := \quad & \{\, \{X,Y\} \in E_T \mid X \neq Q \wedge Y \neq Q\} \cup \{A \cap Q, B \cap Q\} \\ \cup \quad & \{\, \{X, A \cap Q\} \mid \{X,Q\} \in E_T \text{ und } X \in A\} \\ \cup \quad & \{\, \{X, B \cap Q\} \mid \{X,Q\} \in E_T \text{ und } X \in B\} \end{aligned}$$

$c(\{A \cap Q, B \cap S\}) = c((A,B))$, $c(\{X, A \cap Q\}) = c(\{X,Q\})$, $c(\{X, B \cap Q\}) = c(\{X,Q\})$, $c(\{X,Y\})$ bleibt erhalten.
8 Gib T aus.

Während der Berechnung eines Schnittbaumes mit dem Gomory-Hu-Algorithmus (siehe Algorithmus 1) induziert T zu jedem Zeitpunkt eine Partition auf V. Am Ende ist jedes Element der Partition einelementig und T ein Schnittbaum. Die in den einzelnen Iterationsschritten berechneten Schnitte (A,B) kreuzen sich paarweise nicht, denn jeder neue Schnitt verläuft komplett innerhalb eines Elements Q aus T. Anschließend wird Q entlang dieses Schnittes gespalten, so dass der Schnitt von keinem der späteren Schnitte gekreuzt werden kann.

Die Konstruktion eines Schnittbaumes T kann mit jeder Folge von Schnitten (A_i, B_i) die sich paarweise nicht kreuzen analog durchgeführt werden.

Gusfield entwickelte in [Gus90] eine Methode, die einen Gomory-Hu-Baum durch $n-1$ Berechnungen eines minimalen x-y-Schnittes auf dem ursprünglichen Graphen erzeugt. Dadurch werden zwar die Schnittberechnungen aufwendiger, da sie auf einem größeren Graphen durchgeführt werden müssen, dafür spart man sich das aufwendige Berechnen immer neuer Graphen G'. Gusfield zeigt, dass der x-y-Schnitt (A,B) in G' in Schritt 6 durch einen beliebigen minimalen x-y-Schnitt (A',B') in G berechnet werden kann. Zum besseren Verständnis eines Schnittbaumes trägt der Algorithmus von Gusfield nur wenig bei, die Implementierung dagegen ist wesentlich einfacher als die des Algorithmus von Gomory und Hu. Ich habe daher für meine experimentelle Auswertung in Kapitel 5 den Algorithmus von Gusfield implementiert und auf eine detaillierte Beschreibung des Algorithmus verzichtet.

Kapitel 3

Min Schnitt Algorithmen

In diesem Kapitel werde ich untersuchen, in wie weit sich Min Schnitt Algorithmen dazu verwenden lassen, einen minimalen Schnittbaum zu erzeugen.

Der Algorithmus von Stoer und Wagner [StWa94] wird dabei eine besondere Rolle spielen, da er mit seiner Laufzeit von $O(n^2\, log(n) + n\, m)$ schneller ist als der bisher schnellste Algorithmus zum Erstellen eines minimalen Schnittbaumes. Dieser benötigt $n-1$ Flussberechnungen und hat damit eine Laufzeit von $O(n^2\, m\, log(n))$.

3.1 Problemstellung: Min-Schnitt

Gegeben ist ein zusammenhängender, gewichteter Graph $G = (V, E)$ mit Kantengewichten c.
Gesucht ist eine *minimaler, nicht trivialer Schnitt*, also eine Eckenmenge $S \subset V$, $S \neq \emptyset$ so, dass das Gewicht

$$c(S) = \sum_{\substack{\{v,w\}\in E:\\ v\in S, w\in V\setminus S}} c(\{v,w\})$$

minimal wird.

Für den minimalen Schnitt macht es keinen Unterschied, ob eine Kante $\{v, w\}$ im Graphen Gewicht 0 hat oder ob sie nicht existiert. Daher werde ich im Folgenden von einem vollständigen Graphen ausgehen und jeder neu hinzugefügten Kante $\{v, w\}$ das Gewicht $c(\{v, w\}) = 0$ zuordnen.
Auch für das Problem des minimalen Schnittbaumes wird diese Vereinfachung Bestand haben. Jeder Schnitt im Graphen G behält bei der Vervollständigung sein Gewicht und ein Schnittbaum darf ohnehin jedes $\{v, w\} \in V \times V$ als Kante verwenden, unabhängig davon, ob $\{v, w\}$ in E liegt oder nicht.
Für den Fall, dass ein Schnittbaum nur Kanten aus E verwendet, nennt man die zugehörige Schnittbasis *fundamentale Schnittbasis*. Die Suche nach einer minimalen fundamentalen Schnittbasis ist im Allgemeinen *NP*-schwer und wird zum Beispiel in [Sch05] näher erläutert.

3.2 Erste Min-Schnitt-Algorithmen

Ein minimaler Schnitt S darf nicht trivial sein, also trennt er mindestens zwei Ecken; ich nenne sie s und t. Da es keinen leichteren Schnitt, der s und t trennt, im Graphen G geben kann, ist S ein minimaler s-t-Schnitt. In [Bun06] wird folgende Proposition bewiesen:

Proposition 3.1. *Sei D eine minimale Schnittbasis von $G = (V, E)$. Dann enthält D für jedes Eckenpaar $u, v \in V$ einen Schnitt $d \in D$ der die Ecken u und v minimal trennt.*

Damit ist S, oder ein Schnitt mit dem gleichen Gewicht, der ebenfalls s und t trennt, in jedem minimalen Schnittbaum enthalten. Dieser kann mit Hilfe von $n-1$ Flussberechnungen erstellt werden. Also kann auch ein minimaler Schnitt durch $n-1$ Flussberechnungen gefunden werden.

In diesem Fall ist die Bedingung, dass die berechneten Schnitte sich nicht kreuzen dürfen sogar überflüssig. Da der minimale Schnitt auf alle Fälle zwei Knoten des Graphen trennt, genügt es eine Ecke s festzuhalten und dann für alle anderen Ecken t des Graphen einen minimalen s-t-Schnitt zu berechnen. Eine der gewählten Ecken t muss durch den minimalen Schnitt von s getrennt werden und damit ist der entsprechende minimale s-t-Schnitt eine Min-Schnitt des Graphen.

Lange Zeit wurden auch fast ausschließlich Min-Schnitt-Algorithmen entwickelt, die die Dualität von Min-Flow und Max-Cut ausnutzten. Damit waren diese Algorithmen immer gerade so schnell wie $n-1$ Ausführungen des schnellsten Algorithmus zur Flussberechnung. Momentan liegen die schnellsten Flussalgorithmen in $O(n\,m\,log(n))$ [KiRaTa94] (genauere Schranken siehe [RaGo98]).

3.3 Stoer und Wagner

3.3.1 Der Algorithmus

Der Algorithmus von Stoer und Wagner [StWa94] verwendet im Gegensatz zu bisherigen Min-Schnitt-Algorithmen keine Flussberechnung um den minimalen Schnitt zu berechnen. Anstelle dessen benutzt er eine Anordnung der Ecken, die garantiert, dass die letzte Ecke t und die vorletzte Ecke s der Anordnung minimal durch den Eckenschnitt $\{t\}$ voneinander getrennt werden. Die maximale Adjazenzordnung in der folgenden Definition ist eine solche Anordnung.

Definition 3.1 (maximale Adjazenzordnung)**:**
Sei $G = (V, E)$ ein Graph mit Gewichtsfunktion $c : E \to \mathbb{R}^+$. Für $X \subset V$ und $v \in V$ sei

$$c(X, v) := \sum_{x \in X} c(x, v)$$

Eine Ecke $v \in V$, die $c(X, v)$ maximiert, heißt *am stärksten mit X verbunden.* Eine *maximale Adjazenzordnung* ist eine Reihenfolge $A = \{v_1, \ldots, v_n\}$ aller Ecken aus V, für die gilt:

$$v_{i+1} \text{ maximiert } c(\{v_1, \ldots v_i\}, v) \text{ für } v \in V \backslash \{v_1, \ldots, v_i\}$$

Um eine maximale Adjazenzordnung zu erstellen teile ich die Ecken in zwei Mengen, A und $V \backslash A$. Die Menge A enthält die Ecken, die bereits angeordnet sind, $V \backslash A$ den Rest. Am Anfang besteht A nur aus einer einzigen, frei gewählten Startecke v_1. Anschließend wähle ich fortlaufend eine der Ecken $v \in V \backslash A$, die am stärksten mit A verbunden sind, und nehme sie als v_{n+1} in A auf.

Es lässt sich beweisen, dass unter dieser Voraussetzung die letzte und die vorletzte Ecke immer minimal durch einen Eckenschnitt der letzten Ecke voneinander getrennt werden (siehe [StWa94] oder ausführlicher in [Wag06]):

Lemma 3.2. *Sei $G = (V, E)$ ein Graph, c seine Gewichtsfunktion und $A = \{v_1, \ldots, v_n\}$ eine maximale Adjazenzordnung. Der Parameter t bezeichne die letzte Ecke v_n der Anordnung, s die Ecke v_{n-1}. Dann gilt:*

Der Eckenschnitt $\{t\}$ trennt die Ecken s und t minimal.

Eine solche Anordnung aufzustellen ist bei Realisierung mit einem Fibonacci Heap (siehe [FredTar87]) in $O(n\,log(n)+m)$ möglich und damit deutlich schneller als eine Flussberechnung, die aktuell mit dem schnellsten Algorithmus in $O(n\,m\,log(n))$ liegt.

Da im Laufe des Algorithmus von Stoer und Wagner so wie im Algorithmus von Gomory und Hu, Ecken miteinander verschmolzen werden, werde ich im folgenden zwischen Ecken und Knoten unterscheiden. Eine Ecke bezeichnet dabei nach wie vor ein Element $v \in V$, ein Knoten eine Eckenmenge. Die Ecken der Graphen, die aus G durch Verschmelzen einiger Ecken entstanden sind, heißen ab jetzt Knoten. Ein Knoten kann also auch aus einer einelementigen Teilmenge von V bestehen. Der Algorithmus läuft wie folgt:

In $n-1$ MinSchnittPhasen (siehe Algorithmus 2) wird je eine maximale Adjazenzordnung berechnet und der entstandenen minimale Schnitt wird gespeichert. Damit in der nächsten MinSchnittPhase nicht noch einmal die selben Knoten getrennt werden können, werden sie zu einem neuen Knoten verschmolzen. Das Verschmelzen funktioniert dabei analog zum Verschmelzen von Knoten in Kapitel 2. Die beiden Knoten werden zu einem neuen Knoten zusammengefasst und alle Kanten, die einen der beiden Knoten als Endpunkt hatten werden nun mit dem zusammengesetzten Knoten verbunden. Kanten zwischen den verschmolzenen Knoten werden gelöscht.

Algorithmus 2 : MinSchnittPhase(Graph $G_i = (V_i, E_i)$)

```
1 A ← {a} // a kann beliebig gewählt werden
2 t ← a
3 solange A ≠ V_i wiederhole
4 |   bestimme den Knoten v ∈ V_i\A mit c(A, v) maximal und setze
  |   A := A ∪ {v}
5 |   s ← t
6 |   t ← v
7 Speichere (V_i\{t}, {t}) als gefundenen Schnitt.
8 Konstruiere aus G_i den Graphen G_{i+1} durch Verschmelzen der beiden
  Knoten s und t.
```

Algorithmus 3 : MinSchnitt(Graph $G = (V, E)$)

$G_1 \leftarrow G$
für $i = 1$ *bis* $|V| - 1$ **wiederhole**
  MinSchnittPhase (G_i)
Gib den kleinsten der gefundenen Schnitte als minimalen Schnitt aus.

Der Algorithmus beruht damit auf zwei Hauptüberlegungen. Zum einen gilt, dass zwei Knoten im minimalen Schnitt entweder in der gleichen Partitionshälfte liegen oder nicht. Wenn im Laufe einer MinSchnittPhase ein minimaler Schnitt, der zwei Knoten s und t trennt, berechnet wird, gibt es zwei Möglichkeiten. Entweder werden diese beiden Knoten auch vom minimalen Schnitt getrennt, dann hat der gefundene Schnitt bereits minimales Gewicht im gesamten Graphen und der Algorithmus hat sein Ziel erreicht, oder die Knoten können verschmolzen werden, ohne den minimalen Schnitt zu gefährden.

Die zweite Überlegung ist, dass es nicht wichtig ist, welche Knoten in einem Schritt minimal voneinander getrennt werden. Es muss nur dafür gesorgt werden, dass im Laufe des Algorithmus jedes Paar von Knoten mindestens einmal voneinander getrennt wird. Deshalb kann ich in der MinSchnittPhase die maximale Adjazenzordnung verwenden und so auf die aufwendige Flussberechnung verzichten.

Der Algorithmus von Stoer und Wagner (Algorithmus 3) berechnet also $n-1$ minimale Schnitte in immer kleiner werdenden Graphen und der kleinste der berechneten Schnitte ist ein minimaler Schnitt des ursprünglichen Graphen.

3.3.2 Die berechneten Schnitte

Mein Ziel ist es, zu untersuchen, ob diese schnelle Berechnung eines minimalen Schnittes verwendet werden kann, um einen minimalen Schnittbaum zu erzeugen. Ich werde dazu einige Resultate aus [Bun06] einführen und verwenden.

Zunächst fällt auf, dass der Algorithmus von Stoer und Wagner $n-1$ Schnitte berechnet, die zumindest in den jeweiligen Graphen G_i zwei Knoten minimal trennen. Es bleibt also die Frage, ob die berechneten Schnitte tatsächlich einen Schnittbaum bilden und ob die zugehörige Schnittbasis minimal ist. Florentine Bunke beweist dazu in [Bun06] die folgende hilfreiche Proposition.

Proposition 3.3. *Sei D eine unabhängige Menge von Schnitten im Graphen G. Sei d ein weiterer Schnitt, der zwei Ecken separiert, die bisher von keinem der Schnitte in D getrennt werden, dann ist auch $D \cup \{d\}$ eine unabhängige Menge von Schnitten.*

Beweis. Die Schnitte in D seien $S_1, \ldots, S_k$. Angegeben jeweils durch eine Hälfte der Partition. Nach Kapitel 1.1 werden zwei Schnitte wie folgt addiert:

$$S_i + S_j = (S_i \cap S_j) \cup (V \backslash S_i \cap V \backslash S_j)$$

Wenn zwei Ecken u und v von keinem der Schnitte in D getrennt werden, so liegen sie o.B.d.A. in jedem S_i und in keinem $V \backslash S_i$. Damit liegen sie auch in

$(S_{i_1} \cap \ldots \cap S_{i_l})$ und in $S := S_{i_1} + \cdots + S_{i_l}$. Der Schnitt S ist für keine Wahl der i_j der triviale Schnitt, denn die Schnitte in D sind nach Voraussetzung linear unabhängig. $V \backslash S$ ist also nicht leer. Der neue Schnitt d trennt u und v. Es gilt also o.B.d.A. $u \in d$ und $v \in V \backslash d$. Damit ist weiterhin $u \in S + d$, v nun in $V \backslash \{S + d\}$. $S + d$ ist also auch für keine Wahl der i_j der triviale Schnitt. q.e.d.

Mit Hilfe dieses Resultats lässt sich das folgende Lemma beweisen:

Lemma 3.4. *Die vom Algorithmus von Stoer und Wagner berechneten Schnitte bilden eine Schnittbasis.*

Beweis. Der Algorithmus von Stoer und Wagner berechnet $n - 1$ Schnitte. Da eine Schnittbasis aus $n-1$ linear unabhängigen Schnitten besteht, bleibt also zu zeigen, dass die erzeugten Schnitte linear unabhängig sind. Nach Proposition 3.3 müssen dazu für jeden hinzukommenden Schnitt zwei Ecken gefunden werden, die bisher von keinem Schnitt getrennt wurden.

Der erste Schnitt trennt zwei Ecken und damit auch zwei Ecken, die noch von keinem Schnitt getrennt wurden.
Sei S einer der weiteren Schnitte, X der vorletzte und Y der letzte Knoten der maximalen Adjazenzordnung der MinSchnittPhase, in der S entsteht.

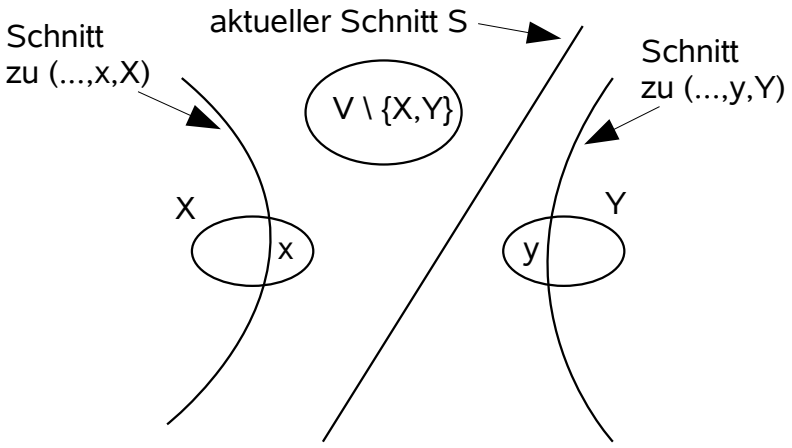


Abbildung 3.1: Beweis der Unabhängigkeit

Behauptung: Der letzte Knoten x, der zu X hinzugekommen ist und der letzte Knoten y, der zu Y hinzugekommen ist, wurden bisher von keinem Schnitt getrennt (siehe Abbildung 3.1). Die maximalen Adjazenzordnungen hatten beim Verschmelzen vom x mit X bzw. von y mit Y die Form $(\ldots, x, X)$ bzw. $(\ldots, y, Y)$.

Jeder Schnitt, der durch den Algorithmus von Stoer und Wagner erzeugt wird hat die Form $(V_i \backslash \{t\}, \{t\})$. Ich werde die Menge $V_i \backslash \{t\}$ *große Menge* und die Menge $\{t\}$ *kleine Menge* des Schnittes nennen, auch wenn $\{t\}$ als zusammengesetzter Knoten in V natürlich mehr Ecken enthalten könnte als $V_i \backslash \{t\}$.

Der Knoten x genau wie der Knoten y war bisher immer in der großen Menge der Schnitte, da sie nie letzter Knoten in einer maximalen Adjazenzordnung waren. Andernfalls wäre x bzw. y schon vorher mit einem anderen Knoten verschmolzen worden und die maximale Adjazenzordnungen $(\ldots, x, X)$ bzw. $(\ldots, y, Y)$ hätte nicht entstehen können.

Falls nun x und/oder y auch wieder zusammengesetzte Knoten sind, so lässt sich rekursiv argumentieren, dass die letzten Knoten, die zu x bzw. y hinzugekommen sind vorher noch nicht getrennt wurden. q.e.d.

Bemerkung Die anderen Ecken aus den zusammengesetzten Knoten X und Y sind als Zeugen für die Unabhängigkeit nicht geeignet. Alle Ecken aus X werden von den Ecken aus $Y\backslash\{y\}$ getrennt, wenn y zu Y hinzugefügt wird. Von y werden die Ecken aus $X\backslash\{x\}$ getrennt, wenn x zu X hinzugefügt wird.

Um die Frage, ob die durch Stoer und Wagner berechnete Schnittbasis durch einen Schnittbaum dargestellt werden kann, zu klären, muss ich untersuchen, ob sich zwei der Schnitte kreuzen. Falls ich das widerlegen kann so liefert das Lemma 2.1 die Existenz eines Schnittbaumes.

Lemma 3.5. *Die von Stoer und Wagner berechneten Schnitte kreuzen sich nicht.*

Beweis. Der erste Schnitt ist ein Eckenschnitt $(V\backslash\{t_1\}, t_1)$ und kann daher von keinem der weiteren Schnitte gekreuzt werden. Die Ecke t_1 liegt entweder in S oder in $V\backslash S$ und damit ist es unmöglich, dass sowohl $S \cap t_1$ als auch $V\backslash S \cap t_1$ nicht leer sind.

Die weiteren Schnitte, die im Algorithmus vorkommen sind immer Eckenschnitte $(V_i\backslash\{t_i\}, t_i)$ des Graphen G_i . Damit ein solcher Schnitt von einem der nächsten Schnitte gekreuzt werden könnte müsste der Knoten t_i wieder geteilt werden. Da dies nicht der Fall ist, kann keiner der entstandenen Schnitte von einem der nachfolgenden Schnitte gekreuzt werden. Es kreuzen sich folglich keine zwei Schnitte. q.e.d.

Damit kann aus den $n-1$ Schnitten des Algorithmus ein Schnittbaum erzeugt werden. Dies geschieht, analog zum Erzeugen eines Schnittbaumes bei Gomory-Hu, parallel zum Erzeugen der Schnitte in $O(m)$ pro Durchlauf der MinSchnittPhase (siehe Algorithmus 1). Da eine MinSchnittPhase einen Aufwand von $O(m + n\,log(n))$ hat, beeinträchtigt das Erstellen des Schnittbaumes die Laufzeit nicht.

Eine Frage, die sich hier direkt stellt ist, ob der Algorithmus von Stoer und Wagner bei geeigneter Wahl des Graphen und geeigneter Durchführung des Algorithmus jede beliebige Baumstruktur erzeugen kann. Schließlich wird in jedem Schritt ein Eckenschnitt im Graphen G_i erzeugt, was auf den ersten Blick nach einer deutliche Einschränkung der möglichen Struktur des Baumes aussieht. Es stellt sich aber schnell heraus, dass folgende Aussage gilt:

Lemma 3.6. *Die Schnitte des Algorithmus von Stoer und Wagner können bei geeigneter Wahl des Graphen jede beliebige Baumstruktur erzeugen.*

Beweisidee: Nach der Wahl eines beliebigen Knoten als Wurzel des Baumes kann der Algorithmus gezwungen werden, zuerst die Knoten der Tiefe 1 mit denen der Tiefe 2 zu verschmelzen und sich so sukzessive bis zur Wurzel hochzuarbeiten. Durch eine geeignete Wahl der Kantengewichte und der Startknoten wird dazu in jedem Schritt die maximale Adjazenzordnung eindeutig vorausbestimmt.

Nun sei der Graph wieder fest vorgegeben. Um zu zeigen, dass sich mit Stoer und Wagner ein minimaler Schnittbaum erstellen lässt, müsste man zeigen, dass bei geeigneter Wahl des Startknotens in den einzelnen MinSchnittPhasen die erzeugten Schnitte minimale Schnitte im Eingabegraphen sind.

Lemma 3.7. *Sei D eine minimale Baum-Schnittbasis des Graphen $G = (V, E)$. Dann gibt es für jeden Schnitt $d \in D$ zwei Ecken u und v, die von d minimal getrennt werden.*

Das Lemma ist eine Folgerung aus zwei Propositionen, die Florentine Bunke in [Bun06] beweist und auf deren Beweis ich hier verzichten werde. Proposition 3.1 habe ich bereits am Anfang dieses Kapitels zitiert, Proposition 3.8 lautet wie folgt:

Proposition 3.8. *Sei D eine Baum-Schnittbasis des Graphen $G = (V, E)$. Dann trennt jeder Schnitt $d \in D$ zwei Knoten aus V, die von keinem der anderen Schnitte aus D getrennt werden.*

Beweis von Lemma 3.7. Sei $d \in D$ ein beliebiger Schnitt. Nach Proposition 3.8 gibt es zwei Ecken, u und v, die von keinem Schnitt $\tilde{d} \in D \backslash \{d\}$ getrennt werden. Also muss d diese Knoten minimal trennen, denn nach Proposition 3.1 trennt mindestens einer der Schnitte aus D die Knoten u und v minimal und keiner der anderen Schnitte kommt dafür in Frage. q.e.d.

Nach Lemma 3.7 ist die Bedingung, dass jeder Schnitt das Knotenpaar, das nur von ihm getrennt wird, minimal trennt, eine notwendige Bedingung dafür, dass der Algorithmus von Stoer und Wagner eine minimale Schnittbasis erzeugt. Sie ist aber auch hinreichend.

Lemma 3.9. *Wenn in einer Baum-Schnittbasis die Zeugen dafür, dass die Schnitte unabhängig sind immer auch minimal vom jeweiligen Schnitt getrennt werden, dann hat die Basis minimales Gewicht.*

Beweis. Gomory-Hu könnte genau die gleiche Schnittbasis berechnen, indem die Zeugen jeweils als nächstes Knotenpaar für eine Schnittberechnung herangezogen werden:
In einer MinSchnittPhase wird ein Schnitt S mit Zeugen u und v berechnet. Da die Zeugen u und v von keinem anderen Schnitt getrennt werden, liegen sie im bis dahin erstellten Gomory-Hu-Baum in der gleichen Komponente. Der Schnitt, für den sie Zeuge sind, schneidet keinen der anderen Schnitte und trennt die Knoten s und t minimal. Daher hätte er auch vom Gomory-Hu-Algorithmus gefunden und verarbeitet werden können. q.e.d.

Wenn also die einzelnen Schnitte jeweils das Knotenpaar (u, v), das nur von ihnen getrennt wird, minimal trennen, so ist die berechnete Basis minimal.

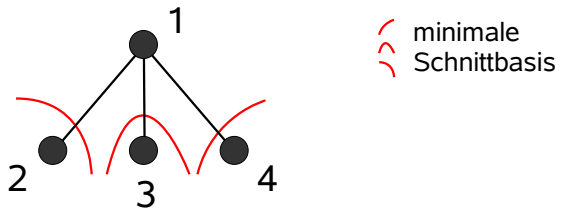


Abbildung 3.2: Gegenbeispiel

Lemma 3.7 führt allerdings zu der Erkenntnis, dass es Instanzen gibt, die der Algorithmus von Stoer und Wagner nicht optimal lösen kann. Ein Beispiel hierfür ist in Abbildung 3.2 gegeben. Selbst bei optimaler Wahl des Startknotens für die maximale Adjazenzordnung in jeder einzelnen MinSchnittPhase, entsteht in mindestens einer MinSchnittPhase ein Schnitt, der keine zwei Ecken des Eingabegraphen minimal trennt und folglich in keiner minimalen Schnittbasis enthalten sein darf.

Die eindeutig bestimmte, minimale Schnittbasis in Abbildung 3.2 hat Gewicht 3. Es können also je zwei Knoten des Graphen durch einen Schnitt mit Gewicht 1 voneinander getrennt werden. Die möglichen Graphenfolgen $(G_i)_{i=1,2,3}$, die im Verlauf des Algorithmus von Stoer und Wagner entstehen können (bis auf Isomorphie), sind in Abbildung 3.3 dargestellt.

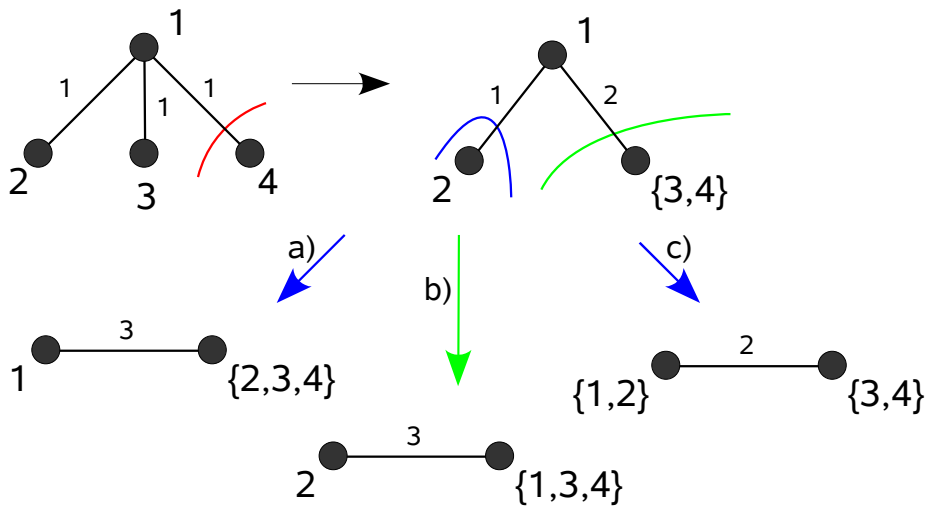


Abbildung 3.3: Graphfolgen

Für $G_1 = G$ führt der Startknoten 1 o.B.d.A zu der Anordnung $\{1, 2, 3, 4\}$ und damit zum Verschmelzen von Knoten 3 mit Knoten 4 und dem zugehörigen minimalen Schnitt $S = \{4\}$. Die Startknoten 2, 3, 4 sind alle isomorph. Startknoten 2 führt o.B.d.A zu der Anordnung $\{2, 1, 3, 4\}$ und damit zum selben Schnitt wie Startknoten 1.

In der zweiten MinSchnittPhase gibt es drei Möglichkeiten in G_2 eine maximale Adjazenzordnungen zu erstellen:

a) Startknoten 1 führt zu der Anordnung $\{1, \{3, 4\}, 2\}$ und damit zu dem minimalen Schnitt $S = \{2\}$ und dem Verschmelzen der Knoten 2 und $\{3, 4\}$.

b) Startknoten 2 führt zu der Anordnung $\{2, 1, \{3, 4\}\}$ und damit bereits zu einem nicht minimalen Schnitt $S = \{\{3, 4\}\}$ mit Gewicht 2.

c) Startknoten $\{3, 4\}$ führt zu der Anordnung $\{\{3, 4\}, 1, 2\}$ und damit zu dem minimalen Schnitt $S = \{2\}$ und dem Verschmelzen der Knoten 1 und 2.

Im dritten Schritt entsteht dann in allen drei Fällen ein Schnitt mit Gewicht > 1 und damit ein Schnitt, der nach Lemma 3.7 nicht in einer minimalen Schnittbasis enthalten sein kann.

3.3.3 Variationen des Algorithmus

Auch Erweiterungen des Algorithmus, die die Laufzeit außer Acht lassen, können die Schnitte nicht in allen Fällen zu einer minimalen Schnittbasis machen. Zwei solche Ideen habe ich untersucht und durch eine gewichtete Version des Gegenbeispiels von oben widerlegt.

Eine Idee ist, alle „äquivalenten“ maximalen Adjazenzordnungen zu einem festen Startknoten in einem Schritt auszuführen, das heißt, falls in der MinSchnittPhase mehrere Knoten gleich stark an die bereits sortierten Knoten gebunden sind, so werden die entstehenden Adjazenzordnungen parallel weitergeführt.

Lemma 3.10. *Schnitte, die in einer MinSchnittPhase des Algorithmus von Stoer und Wagner bei festem Startknoten erzeugt werden können, sind linear unabhängig.*

Beweis. Alle Schnitte die in einer Phase von Stoer und Wagner erzeugt werden können, sind Eckenschnitte des selben Graphen. Da der Startknoten a nie der letzte Knoten einer maximalen Adjazenzordnung sein kann, kann der Eckenschnitt $\{a\}$ nicht entstehen. Es reicht also zu zeigen, dass $k < n_i$ Eckenschnitte in einem Graphen mit $|V_i| = n_i$ Knoten immer linear unabhängig sind.

Für zwei verschiedene Eckenschnitte $S = \{s\} \neq \{t\} = T$ gilt natürlich immer $S \cap T = \emptyset$ und $(V_i \backslash S \cap V_i \backslash T) = (V_i \backslash \{s, t\})$. Also gilt nach Kapitel 1.1:

$$S + T = (V \backslash \{s, t\}) \mathrel{\hat{=}} (\{s, t\}, V \backslash \{s, t\}).$$

Entsprechend gilt für k paarweise verschiedene Eckenschnitte $S_1 = \{s_1\}, \ldots, S_k = \{s_k\}$:

$$S_1 + \cdots + S_k = (\{s_1, \cdots, s_k\}, V_i \backslash \{s_1, \cdots, s_k\}).$$

Die Eckenschnitte sind linear unabhängig, genau dann, wenn $S_1 + \cdots + S_k \neq (V, \emptyset)$, also wenn $\{s_1, \cdots, s_k\} \neq V_i$, bzw. für $k < n_i$. q.e.d.

In dieser Variation des Algorithmus sind damit zwei Fälle möglich. Falls in einer MinSchnittPhase alle möglichen $n_i - 1$ Eckenschnitte erzeugt werden können, so ist eine minimale Schnittbasis im Graphen G_i gefunden. Werden weniger als $n_i - 1$ Schnitte erzeugt, so können diese alle verwendet werden um eine minimale Schnittbasis im Graphen G_i zu erzeugen.

In einer weiteren Variation könnten in jedem Schritt der MinSchnittPhase alle Knoten nacheinander als Startknoten gewählt werden. Da hier der Startknoten variiert, können alle n_i verschiedene Eckenschnitte entstehen. Dann ist aber jede $(n_i - 1)$-elementige Teilmenge dieser Schnitte eine minimale Schnittbasis im Graphen G_i.

In beiden Versionen hätte sich noch die Frage gestellt, wie anschließend Knoten verschmolzen werden. Das könnte zu Problemen führen, da beispielsweise zwei Adjazenzordnungen entstehen können, von der eine mit t, s , die andere mit s, t endet. Dann könnten nur die beiden Knoten s und t verschmolzen werden, wenn der minimale Schnitt nicht gefährdet werden soll. Um die Unabhängigkeit der folgenden Schnitte von den Schnitten {s} und {t} zu gewährleisten muss aber noch ein weiterer Knoten zu s und t hinzugenommen werden, da sonst der Schnitt {s,t} entstehen könnte.

Genauere Überlegungen, wie sich dieses Problem im Allgemeinen lösen ließe lohnen sich nicht, da ich ein Beispiel angeben kann, für das auch bei „Optimierung von Hand“ keine optimale Lösung gefunden werden kann.

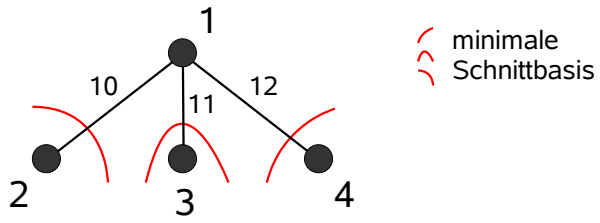


Abbildung 3.4: Gegenbeispiel

Im Graphen aus Abbildung 3.4 ist eine minimale Schnittbasis eindeutig gegeben durch {{2},{3},{4}}. Die möglichen Anordnungen durch unterschiedliche Startknoten und eventuell parallele Ausführung von Wegen ergibt:

- 1, 4, 3, 2
- 2, 1, 4, 3
- 3, 1, 4, 2
- 4, 1, 3, 2

Es können also nur die Eckenschnitte {2} und {3} im ersten Schritt erzeugt werden. Sollen beide Schnitte verwendet werden, so müssen die Knoten 2 und 3 verschmolzen werden. Außerdem muss der Knoten 4 mit diesen beiden Knoten verschmolzen werden damit der Schnitt {2,3} nicht entstehen kann. Also kann der Eckenschnitt {4} nicht mehr gefunden werden.

Wird nur der Schnitt {2} verwendet, so muss Knoten 2 mit Knoten 3 oder 4 verschmolzen werden, womit einer der beider Eckenschnitte {3} oder {4} nicht mehr entstehen kann. Das Gegenbeispiel beweist also, dass sogar unter optimaler Verwendung beider Variationen des Algorithmus die optimale Lösung der Instanz nicht gefunden werden kann.

Fazit Der Algorithmus von Stoer und Wagner eignet sich zwar eventuell als Heuristik für eine minimale Schnittbasis, kann aber nicht zum Erstellen einer optimalen Lösung verwendet werden.

3.4 Ein weiterer Ansatz zum Minimalen Schnitt

Ein weiterer schneller Ansatz zur Berechnung eines minimalen Schnittes in einem Graphen ist der Algorithmus von Hao und Orlin [HO92]. In diesem Algorithmus werden $n-1$ Flussberechnungen zu einem Berechnungsvorgang zusammengefasst. Das senkt die Laufzeit auf $O(n\ m\ log(n^2\ m))$. Auch hier entstehen nach und nach $n-1$ Schnitte des Graphen, die linear unabhängig sind. Jeder einzelne dieser Schnitte trennt eine Eckenmenge X und eine Ecke t minimal voneinander - aber nicht unbedingt zwei Ecken. Damit ergeben sich ähnliche Schwierigkeiten wie beim Algorithmus von Stoer und Wagner.

Kapitel 4

Schnittbaum Heuristiken

Das Erstellen einer minimalen Schnittbasis ist meist nur ein Vorverarbeitungs- oder Teilschritt in anderen Algorithmen. Da auch diese Algorithmen oft nur eine Näherungslösung berechnen, stellt sich die Frage, ob es wirklich notwendig ist, die minimale Schnittbasis zu berechnen oder ob es nicht ausreicht, eine Schnittbasis mit kleinem Gewicht zu erstellen. In diesem Kapitel werde ich daher mögliche Heuristiken zum Erstellen eines Schnittbaumes vorstellen.

Ich suche also nach Algorithmen, die schneller sind als die bekannten minimalen Schnittbaum-Algorithmen, dafür aber auch nur einen Schnittbaum liefern, der nicht unbedingt minimales Gewicht hat.

4.1 Stoer und Wagner als Heuristik

Aus dem letzten Kapitel wissen wir, dass sich der Algorithmus von Stoer und Wagner nicht eignet um eine minimale Schnittbasis zu bestimmen. Er erzeugt aber $n-1$ linear unabhängige Schnitte, die sich paarweise nicht kreuzen und liefert damit die Grundlage für einen Schnittbaum (siehe Kapitel 2). Da der Stoer-Wagner Algorithmus schneller als die bekannten Schnittbaumalgorithmen ist, kann er als Schnittbaumheuristik verwendet werden.

Zunächst sieht die erzeugte Basis auch recht vielversprechend aus. Jeder ihrer Schnitte ist ein minimaler s-t-Schnitt in einem aus G, durch Verschmelzen von Ecken, entstandenen Graphen und trennt damit zwei Knoten ($\hat{=}$ Eckenmengen) s und t minimal. Zudem beinhaltet die Basis mindestens einen Min-Schnitt des Graphen.

In jeder Min-Schnitt-Phase des Algorithmus muss ein Startknoten für die maximale Adjazenzordnung gewählt werden. Ich werde versuchen, diesen Freiheitsgrad auszunutzen, um eine möglichst gute Heuristik zu bekommen. Drei verschiedene Strategien habe ich näher untersucht:

Verwende als Startknoten a immer

1.) den Knoten, der aktuell die meisten inzidenten Kantengewichte hat, also den Knoten $v \in V_i$, der $\sum_{u \in V_i} c(\{v,u\})$ maximiert.

2.) den Knoten, der die wenigsten inzidenten Kantengewichte hat, also den Knoten $v \in V_i$, der $\sum_{u \in V_i} c(\{v, u\})$ minimiert.

3.) $\{s, t\}$ aus dem letzten Schritt, also den gerade erst neu entstandenen Knoten.

Das Ziel der einzelnen Strategien ist das Gesamtgewicht der berechneten Basis zu minimieren. Es lässt sich aber nur schwer abschätzen, welcher Startknoten zu welcher Art von letztem Knoten in der maximalen Adjazenzordnung führen wird. Daher ist die Motivation der verschiedenen Strategien eher wage und es macht Sinn, auch scheinbar gegensätzliche Strategien zu untersuchen:

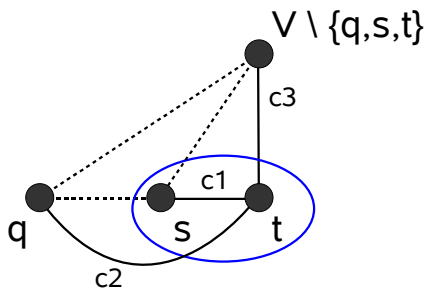


Abbildung 4.1: zu Strategie 2

1.) Bei dieser Strategie ist das Ziel, den Knoten mit den meisten inzidenten Kantengewichten als letzten bzw. vorletzten Knoten in der folgenden maximalen Adjazenzordnung zu vermeiden, da dieser potentiell viel Gewicht zu den berechneten Schnitten beitragen würde.
Bemerkung: Letzter Knoten (ich nenne ihn wieder t) in der maximalen Adjazenzordnung könnte dieser Knoten nur werden, falls es einen weiteren Knoten mit der gleichen Anzahl inzidenter Kantengewichte gibt. Ansonsten wäre $\{s\}$ immer ein kleinerer Schnitt als $\{t\}$ und $\{t\}$ kein minimaler s-t-Schnitt.

2.) Hier ist das Ziel, entlang einer Kante mit großem Gewicht zu verschmelzen, damit diese in keinem weiteren Schnitt auftaucht. Dieses Vorgehen scheint zudem sinnvoll, da das Verschmelzen entlang einer zu leichten Kanten meist einen Fehler nach sich zieht.

   Es seien wie in Abbildung 4.1 s und t zwei verschmolzene Knoten und q der Knoten, der als nächstes mit dem Knoten $\{s, t\}$ verschmolzen wird. Die Gewichte der Kanten seien entsprechend der Abbildung durch c_1, c_2, c_3 gegeben. Nun ergeben sich zwei Fälle für den Schnitt $(S, V \backslash S)$, durch den q mit $\{s, t\}$ verschmolzen wird:

   - $(S, V \backslash S) = (\{s, t\}, V \backslash \{s, t\})$: In diesem Fall muss $c1$ größer sein als $c2 + c3$, da ansonsten $(\{s\}, V \backslash \{s\})$ ein kleinerer Schnitt ist, der s und q trennt.
   - $(S, V \backslash S) = (\{q\}, V \backslash \{q\})$: Hier muss $c1 + c3$ größer sein als $c2$, da ansonsten $(\{q, t\}, V \backslash \{q, t\})$ ein kleinerer Schnitt ist, der s und q trennt.

3.) In dieser Strategie sind für $n > 3$ zumindest die ersten beiden Schnitte tatsächlich minimale s_i-t_i-Schnitte für zwei Ecken s_i und t_i. Die Knoten s_2 und t_2 können noch keine zusammengesetzten Knoten sein, da der einzige zusammengesetzte Knoten $\{s_1, t_1\}$ im zweiten Durchlauf als Startknoten gewählt wurde und somit weder letzter noch vorletzter Knoten der maximalen Adjazenzordnung sein kann. Es geht also darum, möglichst viele Eckenschnitte des Eingabegraphen zu erzeugen.

Leider wird sich in der experimentellen Auswertung im nachfolgenden Kapitel zeigen, dass keine der Strategien zu verlässlich guten Ergebnissen führt. Der Algorithmus von Stoer und Wagner eignet sich also nicht um Schnittbasen mit kleinem Gewicht zu berechnen.

4.2 Weitere Heuristiken

4.2.1 Nützliche Merkmale einer Baum-Schnittbasis

Wenn ich eine Schnittbasis B mit minimalem Gewicht suche, so versuche ich die Zielfunktion

$$c(B) = \sum_{S \in B} \sum_{\substack{\{v,w\} \in E : \\ v \in S,\, w \in V \setminus S}} c(v, w)$$

zu minimieren. Für den Fall einer Baum-Schnittbasis lässt sich die Zielfunktion anders schreiben.

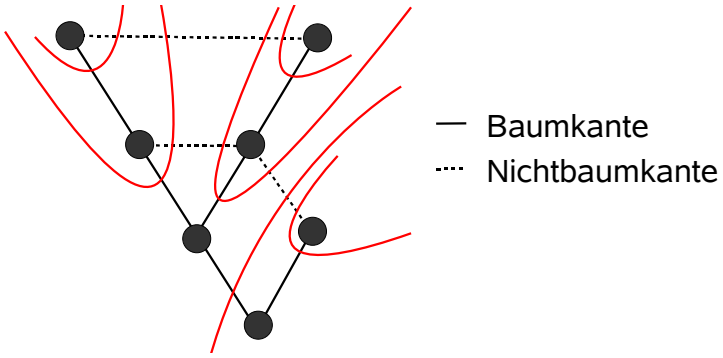


Abbildung 4.2: Baum-Schnittbasis

Lemma 4.1. *Sei B eine Baum-Schnittbasis des Graphen $G = (V, E)$ mit dem zugehörigen aufspannenden Baum $T = (V_T, E_T)$. Dann ist das Gewicht der Basis B gegeben durch:*

$$c(B) = \sum_{\{u,v\} \in E} |p_T(u, v)| \cdot c(\{u, v\}) \tag{4.1}$$

Dabei bezeichnet $p_T(u, v)$ den eindeutig bestimmten Pfad von u nach v in T und $|p_T(u, v)|$ die Länge dieses Pfades.

Beweis. Jede Kante $\{u, v\}$ kommt mindestens in einem Schnitt jeder Schnittbasis vor, da die Ecken u und v mindestens von einem der Schnitte getrennt werden müssen. Eine Kante $\{u, v\}$ gehört zu einem Schnitt, induziert durch die Baumkante $\{w, z\}$, genau dann, wenn der eindeutige Weg in T, der u und v verbindet, durch Entfernen der Kante $\{w, z\}$ zerstört wird (siehe Abbildung 4.2). Also genau dann, wenn $\{w, z\}$ auf $p_T(u, v)$ liegt und diese gilt für $|p_T(u, v)|$ Kanten. q.e.d.

Aus diesem Lemma lassen sich unmittelbar zwei Ansätze für Heuristiken ableiten: Da jede Kante $\{u, v\}$ genau $|p_t(u, v)|$ mal zu dem Gewicht der Basis zählt, ist es sinnvoll, die Wege innerhalb des Baumes T kurz zu halten. Weiterhin gilt, dass eine Baumkante $e \in E_T$ genau einmal zum Gewicht der Basis zählt, jede andere Kante mindestens zweimal. Es wird sich also vermutlich ebenfalls lohnen, einen aufspannenden Baum mit möglichst hohem Gewicht zu wählen.

Erinnerung: Das Gewicht des Schnittbaumes mit den Kanten des betrachteten aufspannenden Baumes ist nicht gleich dem Gewicht des aufspannenden Baumes. Die Gewichtsfunktion des aufspannenden Baumes ist die des Graphen G, die des Schnittbaumes eine andere. Sie ordnet jeder Kante das Gewicht des durch sie induzierten Schnittes zu (siehe Definition 2.1).

4.2.2 Maximaler Baum und kurzer Baum

Auch Anne Schwahn hat in [Sch05] die Heuristiken eines möglichst kurzen bzw. möglichst schweren Baumes betrachtet. Dort allerdings mit dem Unterschied, dass der gefundene Baum eine fundamentale Schnittbasis darstellen soll, also nur Kanten aus dem Graphen benutzen darf.

Definition 4.1:
Ein *maximaler Baum* ist ein aufspannender Baum mit maximalem Gewicht unter allen aufspannenden Bäumen.

Definition 4.2:
Ein *kurzer Baum* ist ein aufspannender Baum T, der

$$L := \sum_{\{u,v\} \in V \times V} |p_T(u, v)| \tag{4.2}$$

minimiert. Also ein Baum, mit möglichst kurzen Wegen innerhalb des Baumes. Dabei spielen die Gewichte der Kanten keine Rolle.

Natürlich müsste man, um das Gewicht der Schnittbasis klein zu halten, an Stelle von L die Summe

$$\sum_{\{u,v\} \in E} |p_T(u, v)| \tag{4.3}$$

minimieren (siehe Gleichung (4.1)). Es wird sich aber zeigen, dass ein kurzer Baum immer auch dieses Kriterium erfüllt.

Ein maximaler Baum kann analog zu einem MST (minimum spanning tree) mit dem Algorithmus von Kruskal in $O(m\ log\, n)$ [Kr56] oder mit dem Algorithmus von Prim in $O(n^2)$ [Pr57] berechnet werden. Da für einen maximalen

Baum Kanten mit möglichst hohem Gewicht benutzt werden, wird der Algorithmus für den Schnittbaum nur Kanten verwenden, die in E enthalten sind. Es ergibt sich also kein Vorteil aus dem Verzicht auf die Fundamentalität für die Heuristik und die Ergebnisse aus [Sch05] gelten unverändert.

Bei der Berechnung eines kurzen Baumes müssten im Falle einer fundamentalen Basis zuerst die kürzesten Wege zwischen allen Eckenpaaren berechnet werden um dann einen geeigneten Teil dieser kürzesten Pfade zu einem Schnittbaum zusammenzusetzen. Die Berechnung der kürzesten Pfade, zum Beispiel mit dem Floyd-Warshall-Algorithmus [Fl62], hätte bereits eine Laufzeit von $O(n^3)$. Wenn aber jeder aufspannende Baum des vollständigen Graphen K_n in Frage kommt, so kann ein kurzer Baum in wesentlich kürzerer Zeit erstellt werden. Ein kurzer Baum ist hier immer ein perfekter Stern, stellt also eine Basis aus Eckenschnitten dar.

Lemma 4.2. *Ein aufspannenden Baum T ist genau dann ein kurzer Baum, wenn er ein perfekter Stern ist. Dabei bezeichnet ein perfekter Stern einen Baum, in dem alle Ecken außer einem ausgezeichneten Mittelpunkt Grad 1 haben.*

Beweis. In T sind genau $n-1$ Eckenpaare durch einen Weg der Länge 1 verbunden, nämlich genau die Eckenpaare, die durch einen Baumkante miteinander verbunden sind. Für alle anderen Paare $u, v \in V$, gilt $|p_T(u,v)| \geq 2$. In einem perfekten Stern $\tilde{T}$ sind alle Eckenpaare durch einen Weg mit einer Länge von maximal 2 miteinander verbunden. Damit minimiert ein perfekter Stern die Summe (4.3).

Umgekehrt gibt es in jedem Baum T, der kein perfekter Stern ist, mindestens ein Eckenpaar $u, v \in V$ mit $|p_T(u,v)| > 2$. Damit gilt:

$$\sum_{\{u,v\} \in V \times V} |p_{\tilde{T}}(u,v)| < \sum_{\{u,v\} \in V \times V} |p_T(u,v)|$$

für alle aufspannenden Bäume T die kein perfekter Stern sind. Die Gleichung (4.2) wird von diesen Bäumen also nicht minimiert. q.e.d.

Durch diesen Beweis wird auch unmittelbar klar, dass die Summe (4.3) durch einen kurzen Baum minimiert wird.

Um eine Schnittbasis zu konstruieren, die durch einen perfekten Stern dargestellt wird, ist folgende Überlegung notwendig. Je $n-1$ Eckenschnitte sind linear unabhängig (siehe Beweis zu Lemma 3.10 im Abschnitt 3.3.3). Damit bilden sie eine Schnittbasis. Ich kann also einen beliebigen der n Eckenschnitte eines Graphen weglassen und erhalte eine Schnittbasis deren zugehöriger aufspannender Baum ein perfekter Stern, also ein kurzer Baum ist. Eine relative Gütegarantie lässt sich ebenfalls angeben.

Lemma 4.3. *Die Heuristik, die eine Schnittbasis mit Hilfe eines kurzen Baumes berechnet, hat eine relative Gütegarantie von 2.*

Beweis. Jede Kante wird beim kurzen Baum in maximal zwei Schnitten gewertet. In mindestens einem muss sie vorkommen, da ihre Endecken nach Proposition 3.1 von mindestens einem Schnitt der Basis getrennt werden müssen. Die Heuristik findet damit auf alle Fälle eine Basis, die maximal das doppelte Gewicht einer optimalen Basis hat. q.e.d.

4.2.3 Maximaler kurzer Baum

Im letzten Abschnitt habe ich die Heuristiken *maximaler Baum* und *kurzer Baum* betrachtet. In diesem Abschnitt möchte ich diese beiden Verfahren zu einem maximalen kurzen Baum kombinieren.

Definition 4.3:
Ein *maximaler kurzer Baum* ist ein kurzer Baum mit maximalem Gewicht unter allen kurzen Bäumen.

Nach Lemma 4.2 muss ich, um einen kurzen Baum zu konstruieren, einen Mittelpunkt wählen. Damit der entstehende Baum maximales Gewicht unter allen kurzen Bäumen hat, wähle ich als Mittelpunkt die Ecke mit den meisten adjazenten Kantengewichten. Die Summe dieser Kantengewichte entspricht dann gerade dem Gewicht des Baumes.

Der gewählte Mittelpunkt entschiedet, welcher der n Eckenschnitte des Graphen nicht in der Schnittbasis enthalten ist. Damit führt ein Mittelpunkt mit maximalen inzidenten Kantengewichten automatisch zu einer Schnittbasis mit minimalem Gewicht unter allen Schnittbasen aus Eckenschnitten.

Die Berechnung eines maximalen kurzen Baumes läuft in $O(n+m)$. In $O(m)$ können die Gewichte der Eckenschnitte, d.h. die Summe der inzidenten Kanten zu jeder Ecke, berechnet werden. In $O(n)$ wird der maximale Eckenschnitt ausgewählt.

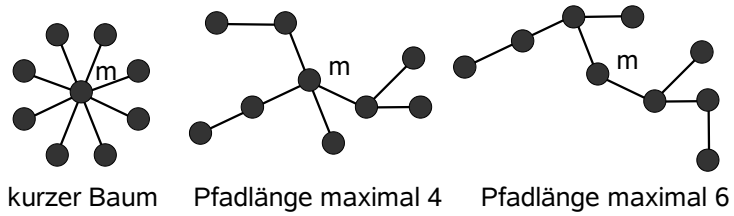


Abbildung 4.3: wachsender Abstand vom Mittelpunkt m

Da ein maximaler kurzer Baum ein perfekter Stern ist, haben alle Ecken den Abstand eins vom Mittelpunkt. Alle Eckenpaare sind damit durch einen Weg der Länge höchstens 2 miteinander verbunden. In einer möglichen Erweiterung der obigen Heuristik können Ecken auch einen Abstand von $2, 3, 4, \ldots$ vom Mittelpunkt haben um einen Baum mit größerem Gewicht zu ermöglichen. Dadurch werden nacheinander Pfade der Länge $4, 6, 8, \ldots$ zugelassen und somit der maximale kurze Baum mehr und mehr in einen maximalen Baum umgewandelt. In dieser allgemeinen Form ist die Erweiterung aber sehr zeitintensiv und damit keine annehmbare Heuristik mehr. Ich werde mich also darauf beschränken, Pfade bis zu einer maximalen Länge von 4 (also einen Abstand von maximal 2 vom Mittelpunkt) durch lokale Optimierung eines bereits berechneten Sternes zu ermöglichen.

Die Idee der lokalen Optimierung eines Baumes T ist das Ersetzen einzelner Baumkanten durch Nichtbaumkanten. Dabei kann eine beliebige Nichtbaumkante $\{u, v\}$ gewählt und zu T hinzugefügt werden. Um den durch $\{u, v\}$ und $p_T(u, v)$ entstandenen Kreis wieder zu beseitigen muss eine beliebige Kante aus

$p_T(u, v)$ aus T entfernt werden.

Anne Schwahn beweist in [Sch05], dass sich das Gewicht des durch die Baumkante $h \in T$ induzierten Schnittes beim Entfernen der Kante e aus T und Hinzufügen von f zu T um

$$c(S_e) - 2 \sum_{g \in S_h \cap S_e} c(g) \quad \forall\, h \in T$$

ändert. Dabei bezeichnet S_e den durch die Kante $e \in T$ induzierten Schnitt. Eine Änderung des Gewichtes $\neq 0$ ergibt sich dabei nur für Kanten $h \in T$ mit $f \in S_h$. Außerdem induziert die Kante f nach dem Kantentausch den gleichen Schnitt wie zuvor die Kante e.

Für eine genauere Begründung der Gewichtsänderung beim Kantentausch in beliebigen Schnittbäumen siehe [Sch05].

Für einen beliebigen Schnittbaum ist es sehr aufwendig, nachzurechnen welcher Kantentausch sich lohnt und welcher nicht. Anne Schwahn löst dieses Problem durch eine recht grobe Abschätzung mit dem Ergebnis, dass sich sicher eine Verbesserung erzielen lässt, falls $c(f) \geq \frac{1}{2} c(S_e)$. Die Abschätzung geht aus der Tatsache hervor, dass für $S_h \cap S_e \neq \emptyset$: $\quad f \in S_h \cap S_e$.

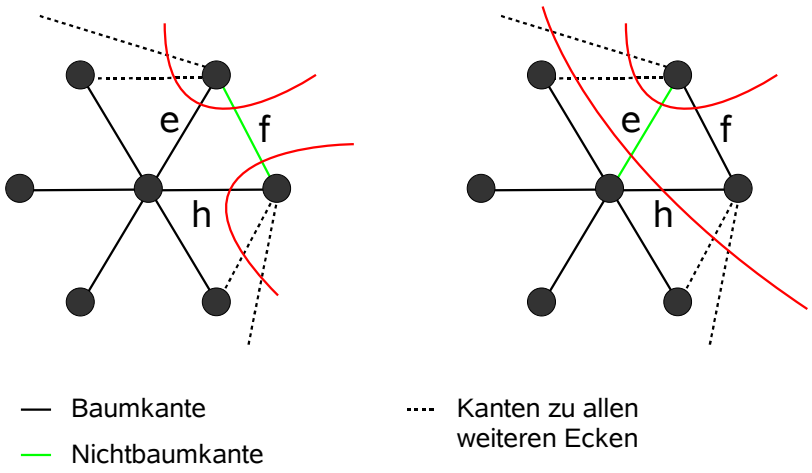


Abbildung 4.4: Kantentausch

Bei einem perfekten Stern als Basis für die Optimierung ist auch die exakte Berechnung des Gewichtsunterschiedes nicht schwer. Es gibt genau eine Kante $h \in T$ mit $f \in S_h$. Für diese Kante wiederum liegt nur f im Schnitt $S_h \cap S_e$ (siehe Abbildung 4.4). Damit lohnt sich ein Kantentausch genau dann, wenn

$$c(S_e) - 2c(f) < 0 \tag{4.4}$$

Die bei Anne Schwahn vorgeschlagene Vereinfachung entspricht hier also der exakten Berechnung.

In einem kurzen Baum ist $|p_t(u, v)| = 2$ für jede Nichtbaumkante $\{u, v\}$. Die Gewichte $c(S_e)$ für $e \in T$ entsprechen gerade den Gewichten der Schnitte

von T und wurden beim Erzeugen des kurzen Baumes bereits berechnet. Damit bleiben für alle Nichtbaumkanten maximal 2 Vergleiche um zu berechnen, ob sich ein Kantentausch lohnt.

Abbildung 4.4 illustriert außerdem, dass die Gleichung (4.4) ebenfalls exakt angibt, ob sich der Kantentausch lohnt, falls f zwei Blattecken u, v mit $|p_T(u, v)| = 2$ verbindet. Ob der Rest des Baumes ein perfekter Stern ist, ist unerheblich.

Bei der Frage, wann es sich lohnt an die durch die Optimierung entstandenen Ecken mit Abstand 1 zum Mittelpunkt mehr als eine Blattecke anzuhängen, ist Abbildung 4.5 hilfreich. Die bereits bestehenden Blattecken seien $v_1, \ldots, v_k$, die Blattecke an e sei w. Dann lohnt sich ein Kantentausch falls:

$$c(S_e) < 2 \cdot (c(f) + \sum_{i=1,\ldots,k} c(v_i, w)) \tag{4.5}$$

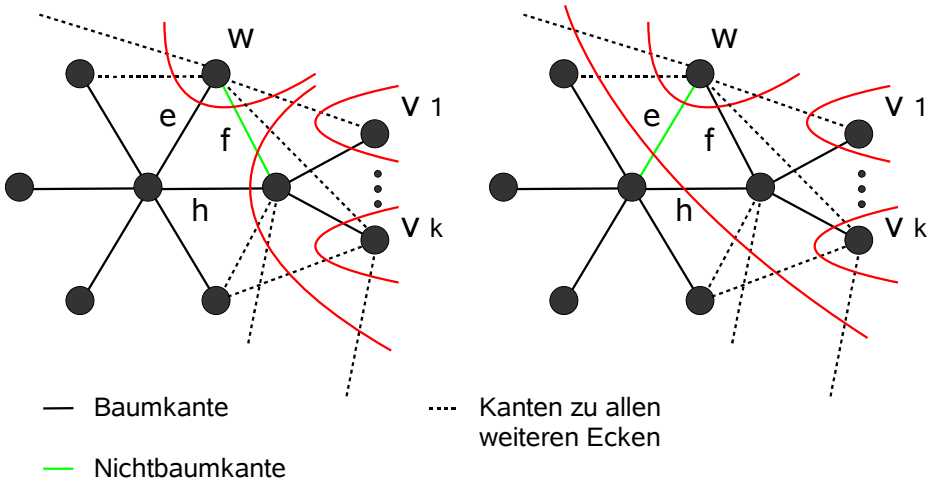


Abbildung 4.5: Kantentausch nach einem Optimierungsschritt

Für die Laufzeit der Heuristik mit Kantentausch und maximal k Blattecken an einer Ecke mit Abstand 1 vom Mittelpunkt ergeben sich damit zwei Möglichkeiten.

a) k wird im Voraus festgelegt, dann ist die Berechnung für ein Kantentausch in $O(1)$ bzw. in $O(k)$ möglich und die Heuristik läuft weiterhin in $O(n+m)$

b) k wird nur durch n beschränkt, dann vergrößert sich die Laufzeit der Berechnung für einen bestimmten Kantentausch auf $O(n)$ und die Gesamtlaufzeit wird $O(n^2 + nm)$

Definition 4.4:
Einen Baum, der aus einem maximalen kurzen Baum durch Optimierung wie oben beschrieben entstanden ist, werde ich im Folgenden einen *optimierten Baum* nennen.

Lemma 4.4. *Die Heuristik, die eine Schnittbasis mit Hilfe eines optimierten Baumes berechnet, hat eine relative Gütegarantie von 2. Die Schranke 2 ist asymptotisch scharf.*

Beweis. Die Gütegarantie von 2 gilt, da sie für jeden kurzen Baum gilt und die Basis des optimierten Baumes kein größeres Gewicht hat als die des maximalen kurzen Baumes aus dem sie erzeugt wurde.

Andererseits zeigt Abbildung 4.6 einen Graphen mit n Ecken und einen zugehörigen maximalen kurzen Baum. Ein minimaler Schnittbaum des Graphen ist der Graph selbst. Der Schnittbaum hat also ein Gewicht von $n-1$. Der maximale kurze Baum dagegen hat Gewicht $(n-3)\cdot 2+2$. Der Approximationsfaktor des kurzen Baumes in diesem Beispiel ist also

$$\frac{(n-3)\cdot 2+2}{n-1} = \frac{2n-4}{n-1} \longrightarrow 2 \qquad \text{für} \quad n \to \infty$$

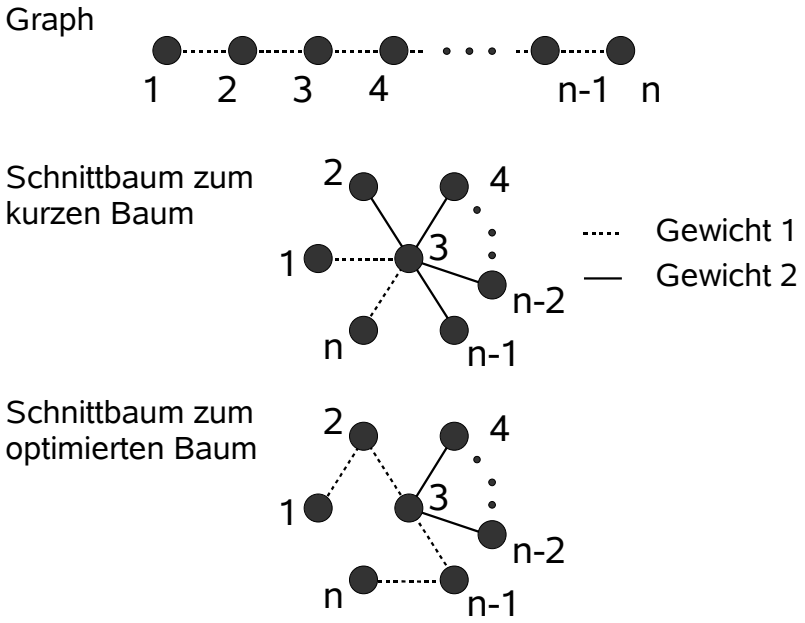


Abbildung 4.6: Gütegarantie 2

Eine Optimierung eines perfekten Sternes durch Ersetzen der Baumkanten e mit der Nichtbaumkante f lohnt sich, falls $c(S_e) < 2\,c(f)$ (Ungleichung (4.4)). In diesem Fall kommen also nur die Kanten $\{1,2\}$ und $\{n,n-1\}$ als Nichtbaumkanten mit zugehörigen Baumkanten $\{1,3\}$ und $\{n,3\}$ in Frage. Im Anschluss an diese Optimierung erfüllt kein Kantenpaar e, f aus Baumkante und Nichtbaumkante die Bedingung (4.5). Der optimierte Baum hat damit den Approximationsfaktor

$$\frac{(n-5)\cdot 2+4}{n-1} = \frac{2n-6}{n-1} \longrightarrow 2 \qquad \text{für} \quad n \to \infty$$

Damit ist bewiesen, dass die Heuristik asymptotisch keinen besseren Approximationsfaktor als 2 hat. q.e.d.

4.2.4 Mehrfacher kurzer Baum

In einzelnen Fällen führt der im letzten Abschnitt beschriebene Optimierungsschritt dazu, dass einer oder sogar mehrere fast gleichwertige Mittelpunkte entstehen. Besonders wenn der Graph einige wenige Ecken mit sehr hohen Graden hat und alle anderen Ecken sehr kleine Grade. Es stellt sich also die Frage, ob es nicht sinnvoll ist, anstelle des Optimierungsschrittes direkt mit mehreren Mittelpunkten zu beginnen.

Dazu lege ich einen Prozentsatz p fest und mache alle Ecken mit adjazenten Kantengewichten von mindestens

$$\text{MinGewicht } + p \cdot (\text{MaxGewicht } - \text{ MinGewicht})$$

zu Mittelpunkten. Den Rest der Ecken verbinde ich jeweils mit dem Mittelpunkt, an den er am stärksten gebunden ist. Abschließend erzeuge ich eine Kante zwischen allen Mittelpunkten und der Ecke mit den meisten adjazenten Kantengewichten.

Algorithmus 4 : MehrfacherKurzerBaum(Graph $G = (V, E)$, c , p)

```
1  Sei V := {v_1, ..., v_n}
2  für  i = 1 bis |V| wiederhole
3  |   AdjGewichte[i] ← Σ_{{u,v_i}∈E} c({u, v_i})
4  MinGewicht ← min_i AdjGewichte[i]
5  MaxGewicht ← max_i AdjGewichte[i]
6  Grenzgewicht ← MinGewicht + p · (MaxGewicht − MinGewicht)
7  Mittelpunkte ← {v_i ∈ V : AdjGewichte[i] ≥ Grenzgewicht}
8  für  v_i ∉ Mittelpunkte wiederhole
9  |   verbinde v_i zu dem Mittelpunkt, zu dem es am stärksten verbunden
   |   ist.
10 für  v_i ∈ Mittelpunkte wiederhole
11 |   verbinde v_i zu der Ecke mit den meisten adjazenten Kantengewichten.
12 Berechne für alle Mittelpunkte den Schnitt, der durch die Kante zum
   innersten Mittelpunkt induziert wird.
```

Leider benötigt die Berechnung des Gewichtes eines Schnittes der durch eine Kante zwischen zwei Mittelpunkten induziert wird einen Aufwand von $O(n^2)$. Im schlimmsten Fall führt das also zu einem Aufwand von $O(n^3)$ und die Heuristik ist nicht besonders schnell. Dieser Grenzfall tritt aber kaum auf und kann auch dadurch abgefangen werden, dass die maximale Anzahl der Mittelpunktecken beschränkt wird.

Der Ansatz hat den Vorteil, dass keine falschen Mittelpunkte ausgewählt werden. Beim optimierten Baum dagegen kann frühzeitig eine „falsche“ Ecke umgehängt werden und damit nicht mehr als Mittelpunkt zur Verfügung stehen.

Kapitel 5

Experimentelle Auswertung

5.1 Die Tests

Im Anschluss an die theoretische Beschreibung der Schnittbaumheuristiken werde ich sie nun an ausgewählten Graphen testen. In der Testumgebung kommen AMD Opterons ($\geq$ 2.2 GHz) mit mindestens acht Gigabyte Speicher unter Linux Version 2.6 zum Einsatz. Der großzügige Speicherausbau gewährleistet dass der gesamte Test im physikalischen Arbeitsspeicher ablaufen kann. Ich habe alle Algorithmen in Java implementiert und mit der Java 1.5 Standard Edition ausgeführt.

Die ersten Testgraphen habe ich mit dem „Inet-3.0“-Generator erzeugt (eine ausführliche Beschreibung ist unter [Inet-3.0] zu finden). Dieser Generator wurde entwickelt um Graphen zu erzeugen, die die Internet Topologie simulieren. Er bekommt zwei Parameter übergeben, die direkte Auswirkungen auf den erzeugten Graphen haben. Die übrigen Parameter werde ich nicht weiter berücksichtigen, da sie nur die Einbettung betreffen. Der Parameter n bezeichnet die Anzahl der erzeugten Ecken, d den Anteil der Ecken vom Grad 1. Ich habe $n \in \{3500, 4000, 5000, 6000\}$ und $d \in \{0.1, 0.3, 0.5\}$ gewählt. (Die Mindestanzahl an Ecken für den Generator beträgt 3037).

Um die Heuristiken in den Tabellen und Zeichnungen zu referenzieren werde ich folgende Abkürzungen benutzen:

MaKB	$\hat{=}$	Maximaler kurzer Baum
OBk	$\hat{=}$	Optimierter Baum mit Parameter k
MeBp	$\hat{=}$	Mehrfacher kurzer Baum mit Faktor $0, p$
MaB	$\hat{=}$	Maximaler Baum
SWi	$\hat{=}$	Stoer und Wagner mit Strategie i

Das Gewicht der berechneten Schnittbäume im Verhältnis zum Gewicht des jeweiligen minimalen Schnittbaumes ist in Tabelle 5.1 und in Tabelle 5.2 zu sehen. Es fällt auf, dass der maximale Baum ausschließlich Schnittbasen erzeugt, die mindestens das 6.5-fache Gewicht der optimalen Lösung haben. Der Algorithmus von Stoer und Wagner erzielt Approximationsfaktoren zwischen 3 und 5, schneidet also auch sehr schlecht ab. Die übrigen Algorithmen dagegen erzielen konstant gute Werte, wobei die optimierten Bäume am besten abschneiden.

n	d	MaKB	OB1	OB50	MeB1	MeB50
3500	0.1	1.033	1.019	1.006	1.033	1.091
3500	0.3	1.095	1.066	1.012	1.095	1.133
3500	0.5	1.181	1.135	1.028	1.181	1.208
4000	0.1	1.032	1.017	1.006	1.032	1.101
4000	0.3	1.094	1.066	1.013	1.094	1.136
4000	0.5	1.173	1.131	1.026	1.173	1.193
5000	0.1	1.032	1.019	1.006	1.032	1.067
5000	0.3	1.090	1.063	1.011	1.090	1.124
5000	0.5	1.167	1.126	1.029	1.167	1.203
6000	0.1	1.029	1.017	1.005	1.029	1.077
6000	0.3	1.087	1.061	1.011	1.087	1.136
6000	0.5	1.160	1.122	1.028	1.160	1.210

Tabelle 5.1: INet-Graphen: c(Schnittbaum)/c(minimaler Schnittbaum)

n	d	SW1	SW2	SW3	MaB
3500	0.1	3.765	3.785	3.822	12.718
3500	0.3	4.140	4.153	4.215	8.524
3500	0.5	4.606	4.423	4.689	8.073
4000	0.1	3.958	3.965	3.876	12.036
4000	0.3	4.405	4.287	4.358	9.095
4000	0.5	4.631	4.628	4.685	6.587
5000	0.1	4.173	4.110	4.183	11.869
5000	0.3	4.546	4.511	4.499	9.468
5000	0.5	4.863	4.823	4.893	7.213
6000	0.1	4.294	4.302	4.320	12.975
6000	0.3	4.665	4.648	4.692	8.741
6000	0.5	5.086	5.006	5.017	9.979

Tabelle 5.2: INet-Graphen: c(Schnittbaum)/c(minimaler Schnittbaum)

Vor allem bei intensiver Optimierung (OB50) haben alle berechneten Basen maximal das 1.03-fache Gewicht der minimalen Schnittbasis. Zudem lässt sich beobachten, dass der mehrfache kurze Baum mit Faktor $p = 0.1$ exakt die gleichen Ergebnisse wie der maximale kurze Baum liefert. Das heißt, dass es hier nur eine einzige Ecke gibt, die einen, im Vergleich zu allen anderen Ecken, extrem hohen Grad hat.

In Abbildung 5.1 sind die Werte für 5000 Ecken dargestellt. Die Kurven für die Graphen mit 3500, 4000 und 6000 Ecken verlaufen fast identisch.

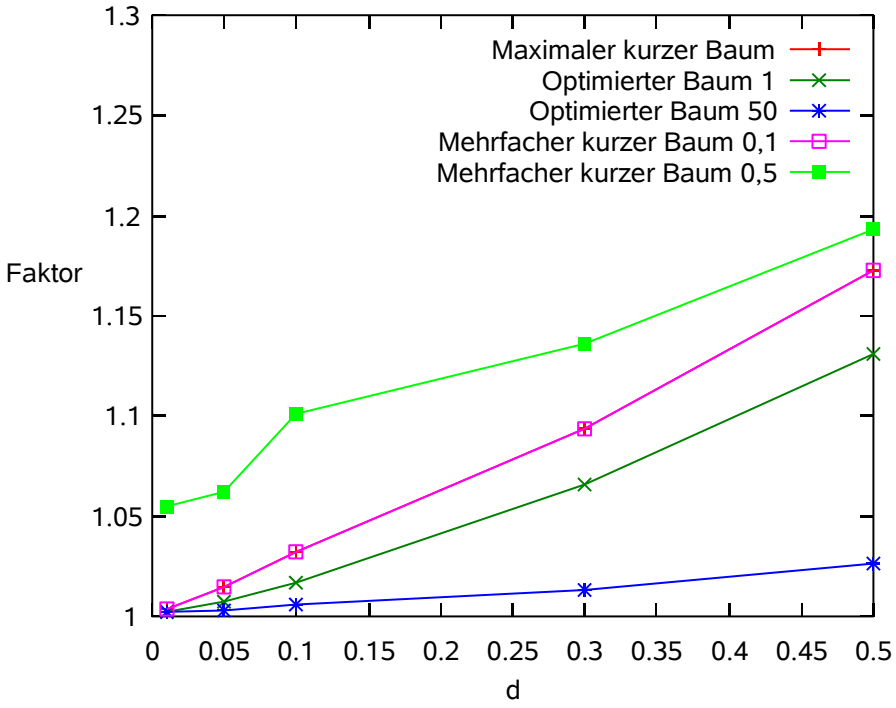


Abbildung 5.1: Grafik zum INet Graphen

Als zweite Graphklasse werde ich Teile des Straßennetzes rund um Karlsruhe betrachten. In Graphen dieser Art sind Ergebnisse von Schnittbaumalgorithmen aus mehreren Gründen interessant. Sie geben Informationen über den Zusammenhang des Straßennetzes und sind natürlich die klassischen Graphen für TSP-Probleme.

Der Ausgangsgraph für meine Tests enthält knapp 50.000 Ecken und etwa 125.000 Kanten. Zur Untersuchung habe ich ihn schachbrettartig in 16 geographisch gleich große Teilgraphen aufgeteilt.

Der optimierte Baum schneidet auch hier am besten ab (siehe Tabelle 5.3). Der Algorithmus von Stoer und Wagner erreicht nur selten eine 1.5-Approximation und bleibt damit deutlich unter den Ergebnissen des maximalen kurzen Baumes. Der maximale Baum berechnet hier zwar bessere Schnittbäume als auf den iNet-Graphen, liefert aber nach wie vor die schlechtesten Ergebnisse.

n	m	MaKB	OB1	OB50	MeB1	MeB50
2967	7328	1.196	1.095	1.083	1.198	1.125
4455	11850	1.124	1.059	1.054	1.127	1.071
2706	6652	1.195	1.093	1.082	1.195	1.115
2192	5408	1.182	1.087	1.076	1.182	1.114
1409	3336	1.223	1.120	1.109	1.262	1.168
5898	15052	1.140	1.064	1.058	1.144	1.090
3214	7856	1.191	1.086	1.074	1.241	1.105
2616	6632	1.152	1.068	1.062	1.154	1.325
3011	7394	1.185	1.095	1.087	1.185	1.126
3004	7730	1.196	1.093	1.083	1.252	1.122
2966	7360	1.171	1.074	1.064	1.171	1.094
3105	8046	1.136	1.056	1.049	1.136	1.316
3146	7496	1.217	1.109	1.097	1.217	1.146
3710	9252	1.162	1.072	1.065	1.163	1.098
2206	5340	1.190	1.085	1.075	1.190	1.116
3056	7524	1.193	1.086	1.075	1.194	1.109

Tabelle 5.3: Daten zum Karlsruhegraph

n	m	MaB	SW1	SW2	SW3
2967	7328	1.997	1.673	1.612	1.610
4455	11850	2.200	1.534	1.514	1.506
2706	6652	1.839	1.543	1.542	1.602
2192	5408	1.962	1.521	1.547	1.567
1409	3336	1.656	1.512	1.572	1.545
5898	15052	2.468	1.592	1.600	1.612
3214	7856	1.927	1.576	1.616	1.608
2616	6632	1.941	1.500	1.514	1.523
3011	7394	1.771	1.566	1.547	1.497
3004	7730	1.587	1.593	1.609	1.553
2966	7360	1.936	1.583	1.615	1.560
3105	8046	1.902	1.540	1.536	1.532
3146	7496	2.044	1.625	1.578	1.631
3710	9252	1.953	1.611	1.590	1.609
2206	5340	1.719	1.595	1.620	1.584
3056	7524	2.024	1.629	1.645	1.656

Tabelle 5.4: Daten zum Karlsruhegraph

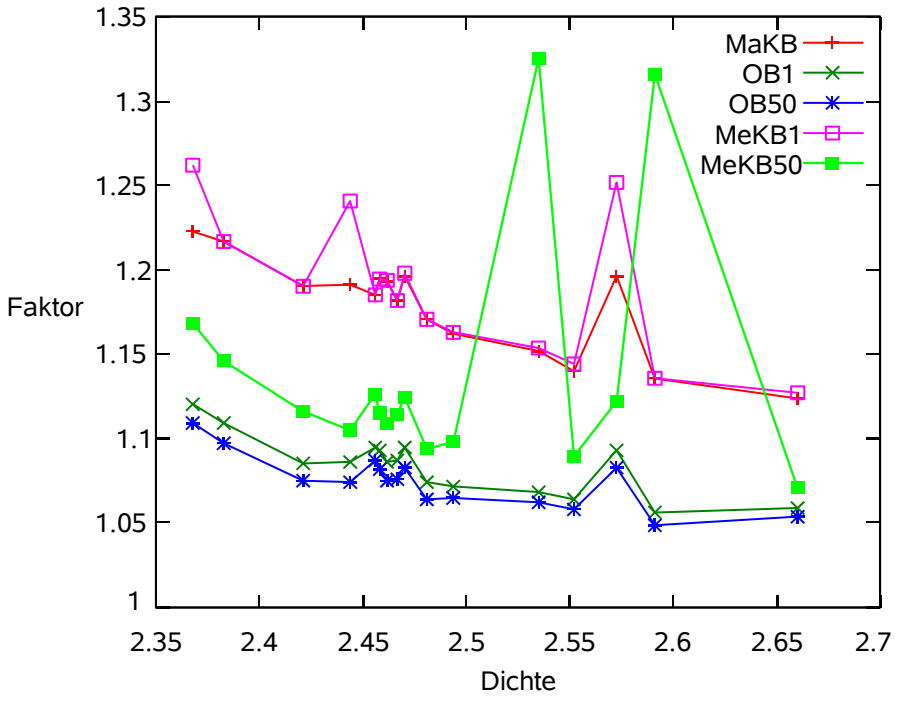


Abbildung 5.2: Grafik zum Karlsruhegraph

Die 16 getesteten Teile des Straßengraphen haben zwischen 1400 und 6000 Ecken. Um zu zeigen, dass sich die Heuristiken in größeren Graphen nicht anders verhalten, habe ich zusätzlich einen Graphen mit ca. 10 000 Ecken untersucht.

n	m	MaKB	OB1	OB50	MeB1	MeB50
9968	25648	1.136	1.061	1.055	1.140	1.307

MaB	SW1	SW2	SW3
3.135	1.601	1.593	1.604

Auf Grund der schlechten Ergebnisse des Algorithmus von Stoer und Wagner und seiner im Vergleich zu den anderen Heuristiken langen Laufzeit, habe ich in allen weiteren Graphen auf ihn verzichtet.

Die dritte von mir betrachtete Graphklasse beinhaltet Random Graphen. Sie sind nach dem Erdos-Renyi-Modell und nach dem Scale-Free-Modell mit dem Programm Pajek erstellt [Pajek]. Ich habe jeweils Graphen mit 100, 1000 bzw. 5000 Ecken betrachtet mit einem durchschnittlichen Eckengrad von 5, 10 und 50; 10, 50 und 100 bzw. 50, 100 und 200.

In allen 9 von mir getesteten Erdos-Renyi-Graphen war der minimale Schnittbaum bereits ein perfekter Stern, daher berechneten die Heuristiken maximaler kurzer Baum und optimierter Baum das optimale Ergebnis. Die anderen Heuristiken schnitten unterschiedlich ab.

n	deg	MaKB	OB1	OB50	MeB1	MeB50	MaB
100	5	1.004	1.000	1.000	1.121	1.273	2.610
100	10	1.000	1.000	1.000	1.119	1.476	3.127
100	50	1.000	1.000	1.000	1.169	1.342	1.448
1000	10	1.000	1.000	1.000	1.053	1.505	4.829
1000	50	1.000	1.000	1.000	1.000	1.768	3.924
1000	100	1.000	1.000	1.000	1.631	1.566	2.671
5000	50	1.000	1.000	1.000	1.147	1.514	6.841
5000	100	1.000	1.000	1.000	1.310	1.547	6.035
5000	200	1.000	1.000	1.000	1.890	1.417	4.499

Tabelle 5.5: Erdos-Renyi

Für die Scale-Free-Graphen habe ich jeweils einen initialen Erdos-Renyi-Graphen mit 10 Ecken und einem durchschnittlichen Eckengrad von 0.5 verwendet. Der Parameter α, der angibt wie sehr eine neue Kante bevorzugt zwischen Ecken mit hohem Grad entsteht, ist in allen Graphen 0.6. Dieser Wert wird üblicherweise in wissenschaftlichen Netzwerken erreicht.

Aus Tabelle 5.6 geht hervor, dass der optimierte Baum auch in diesen Random-Graphen fast den optimalen Schnittbaum berechnet. In einem Fall erreicht der optimierte Baum mit $k = 1$ bereits die optimale Lösung, der optimierte Baum mit $k = 50$ schafft das sogar in 5 Fällen. Die weiteren 1.000-Werte in der Tabelle entstehen durch die Rundung auf 3 Nachkommastellen.

n	deg	MaKB	OB1	OB50	MeB1	MeB50	MaB
100	5	1.035	1.010	1.000	1.035	1.249	1.316
100	10	1.016	1.003	1.002	1.016	1.310	1.723
100	50	1.001	1.000	1.000	1.001	1.307	1.438
1000	10	1.016	1.004	1.001	1.016	1.479	2.811
1000	50	1.001	1.000	1.000	1.001	1.525	2.202
1000	100	1.000	1.000	1.000	1.206	1.538	1.765
5000	50	1.001	1.000	1.000	1.001	1.619	3.384
5000	100	1.000	1.000	1.000	1.228	1.625	2.316
5000	200	1.000	1.000	1.000	2.651	2.709	3.896

Tabelle 5.6: Scale-Free

Abschließend habe ich die Heuristiken noch an drei weiteren realen Graphen getestet. Der erste Graph (DW3N) ist die 3 Nachbarschaft von Prof. Dorothea Wagner, entnommen aus der DBLP 2003 der Universität Trier. Der zweite Graph (MailInf) beschreibt den internen e-mail Verkehr der Mitarbeiter der Informatik Fakultät der Universität Karlsruhe und der dritte Graph (KarNet) das soziale Netzwerk innerhalb eines Karatevereins. Es ist deutlich zu erkennen, dass der optimierte Baum auch hier sehr gute Resultate erzielt. Nur im sozialen Netzwerk ist der mehrfache Baum mit Parameter 0.1 noch etwas besser. Das Netzwerk entstand zu einem Zeitpunkt, zu dem der Verein sich auf Grund von finanziellen Differenzen in zwei Lager spaltete, eines um den Trainer und eines um den Manager. Es ist also nicht weiter verwunderlich, dass der minimale Schnittbaum hier zwei Ecken mit sehr hohem Grad und ansonsten fast nur Ecken vom Grad eins besitzt und der mehrfache Baum mit kleinem p hier be-

sonders gut abschneidet. Die genauen Ergebnisse sind in der folgenden Tabelle dargestellt:

	n	m	MaKB	OB1	OB50	MeB1	MeB50	MaB
DW3N	6481	34521	1.049	1.038	1.030	1.055	1.424	3.359
MailInf	701	4698	1.061	1.046	1.031	1.061	1.449	1.853
KarNet	34	77	1.062	1.054	1.054	1.046	1.069	1.369

Zu der tatsächlichen Laufzeit der Algorithmen lässt sich festhalten, dass der Algorithmus von Gusfield für die großen Graphen mit ca. 5000 Ecken je nach Struktur des Graphen zwischen einer und 33 Stunden lief, die Heuristiken maximaler kurzer Baum und optimierter Baum dagegen immer unter einer Sekunde blieben. Der Algorithmus von Stoer und Wagner lag mit seiner Laufzeit nur knapp unter der des optimalen Algorithmus, die Heuristiken maximaler Baum und mehrfacher Baum blieben meist bei wenigen Sekunden bis Minuten. Die erfolgreiche Heuristik des optimierten Baumes ist also zusätzlich äußerst schnell.

Zu beachten ist, dass der maximale Baum in fast allen Testgraphen von vorne herein im Nachteil war, da alle Graphen bis auf MailInf und DW3N eine konstante Gewichtsfunktion hatten. Die Maximaler-Baum-Heuristik schneidet etwas besser ab, wenn die Graphen eine Gewichtsfunktion mit großer Gewichtsspanne haben. Aber auch dort bleiben die Ergebnisse schlechter als die des optimierten Baumes.

Gute Ergebnisse liefert die Maximaler-Baum-Heuristik in Graphen die bereits fast Baumstruktur haben. Ist der Graph tatsächlich ein Baum, so ist der maximale Baum gerade der Graph selbst und liefert die optimale Schnittbasis. Der kurze Baum sowie der optimierte Baum benutzen hier unnötig viele Kanten zweimal, die nur ein einziges mal in der Schnittbasis verwendet werden müssten. Wenn also in einem Graphen $G = (V, E)$ mit $|E| = m$ und $|V| = n$, m nicht viel größer ist als n, so sollte die Maximaler-Baum-Heuristik verwendet werden.

5.2 Abschließende Bemerkungen

Abschließend lässt sich sagen, dass die Heuristik *optimierter Baum* sich hervorragend eignet um Schnittbasen mit kleinem Gewicht zu berechnen. Mit der Gütegarantie von Faktor 2 ist sie beweisbar gut und im letzten Abschnitt habe ich gezeigt, dass sie auf den getesteten Graphen erheblich besser abschneidet.

Falls erwünscht könnten die Optimierungsschritte auch noch verfeinert werden. Zum Beispiel lohnt es sich wahrscheinlich vorab zu berechnen, wie viel Gewicht gespart wird, durch Ecken, die an eine bestimmte Ecke angehängt werden. Dann könnten zuerst Ecken umgehängt werden, die an die selbe Ecke angehängt werden und besonders viel Gewichtsvorteil bringen. Damit wäre verhindert, dass „falsche“ Ecken umgehängt werden und dann nicht mehr als Ecken der Tiefe 1 zur Verfügung stehen.

Der Algorithmus von Stoer und Wagner dagegen eignet sich für die hier untersuchten Zwecke nicht. Es mag hilfreich sein, zu wissen, dass er bei der Berechnung eines minimalen Schnittes auch automatisch eine Schnittbasis liefert – als Heuristik für einen minimalen Schnittbaum eignet er sich aber nicht.

Literaturverzeichnis

[FoFu56] L.R. Ford und D.R. Fulkerson:
Maximal flow through a network
Canadian Journal of Mathematics 8:399-404 1956

[Kr56] J. Kruskal:
On the shortest spanning subtree of a graph and the traveling salesman problem.
Proceedings of the American Mathematical Society 7:48-50 1956

[Pr57] R. Prim:
Shortest connection networks and some generalizations
Bell System Techical Journal 36:1389-1401 1957

[GH61] R.E. Gomory und T.C. Hu:
Multi-Terminal Network Flows
Journal of SIAM 9:551-570 1961

[Fl62] R.W. Floyd:
Algorithm 97: Shortest path
Communications of the ACM 5:345 1962

[Gib85] Alan Gibbons:
Algorithmic Graph Theorie
Cambridge University Press 1985

[FredTar87] M. L. Fredman und R.E. Tarjan:
Fibonacci heaps and their uses in improved network optimization algorithms
Journal of the ACM 34:596-615 1987

[Gus90] D. Gusfield:
Very simple methods for all pairs network flow analysis
SIAM Journal of Computing 19:143-155 1990

[HO92] J. Hao und J.B. Orlin:
A Faster Algorithm for Finding the Minimum Cut in a Graph
In Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms 165-174 1992

[StWa94] M. Stoer und F. Wagner:
A Simple Min Cut Algorithm
Proceedings of the 1994 European Symposium on Algorithms,
Lecture Notes In Computer Science 855:141-147 1994

[KiRaTa94] V. King, S. Rao, R.E. Tarjan:
A faster deterministic maximum flow algorithm
Journal of Algorithms 17:447-474 1994

[RaGo98] S. Rao A.V. Goldberg:
Beyond the flow decomposition barrier
Journal of the ACM 45:783-797 1998

[GoTs01] A.V. Goldberg und K. Tsioutsiouliklis:
Cut Tree Algorithms: An Experimental Study
Journal of Algorithms 38:51-83 2001

[FlTaTs03] G.W. Flake, R.E. Tarjan und K. Tsioutsiouliklis:
Graph Clustering and Minimum Cut Trees
Internet Mathematics vol 1, num 4:385-408 2003

[Sch05] Anne Margarete Schwahn:
Minimum Fundamental Cut Basis Problem
Diplomarbeit an der TU Kaiserslautern 2005

[Bun06] Florentine Bunke:
Circuit Bases Problems in Binary Matroids
Dissertation an der TU Kaiserslautern 2006

[Wag06] Dorothea Wagner:
Skript zur Vorlesung Algorithmentechnik
`http://i11www.ira.uka.de/teaching/WS_0506/algotech/skript/Skript.pdf`

[Inet-3.0] J. Winick und S. Jamin:
Inet-3.0: Internet Topology Generator
`http://topology.eecs.umich.edu/inet`

[Pajek] V. Batagelj und A. Mrvar
Program for Large Network Analysis
`http://vlado.fmf.uni-lj.si/pub/networks/pajek`