technische universität
dortmund

Baumzerlegungsbasierte Algorithmen für das Steinerbaumproblem

Adalat Jabrayilov

Algorithm Engineering Report
TR15-1-002
Dezember 2015
ISSN 1864-4503

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

technische universität
dortmund

Diplomarbeit

Baumzerlegungsbasierte Algorithmen für das Steinerbaumproblem

Adalat Jabrayilov

12. Mai 2015

Betreuer:
Prof. Dr. Petra Mutzel
Dipl.-Inf. Denis Kurz

Fakultät für Informatik
Algorithm Engineering (Ls11)
Technische Universität Dortmund
http://ls11-www.cs.tu-dortmund.de

Bu diplom işi qardaşim Tural Cäbrayilovun (03.09.1986 - 22.09.2002) äziz xatiräsinä ithaf olunur.

Diese Diplomarbeit ist meinem Bruder Tural Jabrayilov (03.09.1986 - 22.09.2002) gewidmet.

Zusammenfassung

In dieser Diplomarbeit wird ein baumzerlegungsbasierter Algorithmus für das Steinerbaumproblem vorgestellt. Der Algorithmus ist eine Heuristik mit Laufzeit $F + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$, wobei V die Knotenmenge des Graphen, tw die Baumweite einer Baumzerlegung des Graphen und F eine Zufallsvariable mit Erwartungswert $O\left(|V| \cdot 2^{tw} \cdot tw\right)$ ist. Dabei ist die Wahrscheinlichkeit, dass F größer als $|V| \cdot 2^{tw} \cdot tw^3$ ist, ist kleiner als $2^{-|V| \cdot tw^2}$. Die Heuristik wurde implementiert und auf bekannten Benchmark-Instanzen getestet. Die größte beobachtete Güte bei diesen Tests war 1.19.

Inhaltsverzeichnis

1 Einführung **1**

1.1 Motivation und Hintergrund . . . 1

1.2 Aufbau der Arbeit . . . 2

2 Grundlagen **3**

2.1 Graph . . . 3

2.2 Steinerbaumproblem . . . 4

2.3 Baumzerlegung . . . 5

2.4 Notationen . . . 7

3 Algorithmus **9**

3.1 Vorüberlegungen . . . 9

3.1.1 Zulässige Knotenteilmenge . . . 11

3.2 Teillösungen . . . 14

3.3 Dynamisches Programm für einen Spezialfall . . . 16

3.3.1 Ein Beispiel . . . 20

3.3.2 Schnelle Verkettung . . . 25

3.4 Relevante Pfade . . . 27

3.4.1 Trick mit künstlicher Kante . . . 27

3.4.2 Relevante Pfade . . . 29

3.4.3 Traceback für relevante Pfade . . . 35

3.5 Dynamisches Programm für allgemeine Graphen . . . 38

3.5.1 Algorithmus für das Steinerbaumproblem . . . 38

3.6 Analyse . . . 40

4 Evaluierung **47**

4.1 Implementierung . . . 47

4.1.1 Join-Tree Konstruktion . . . 47

4.1.2 Einige Abweichungen . . . 48

4.1.3 Das Programm . . . 49

4.2 Tests . . . 50

4.2.1 Zusammenhang: Baumweite $tw \leftrightarrow$ Güte r 52
4.2.2 Zusammenhang: Knotenanzahl $|V| \leftrightarrow$ Güte r 54
4.2.3 Zusammenhang: $\frac{|T|}{|V|} \leftrightarrow$ Güte r 56
4.2.4 Fazit 58

5 Zusammenfassung und Ausblick **59**
5.1 Ein weiterer Lösungsansatz 59

A Testergebnisse **61**

Abbildungsverzeichnis **72**

Literaturverzeichnis **74**

Erklärung **74**

Kapitel 1

Einführung

1.1 Motivation und Hintergrund

Diese Diplomarbeit befasst sich mit dem Steinerbaumproblem. Um dieses Problem zu motivieren, betrachten wir Internet-Multiplayer-Spiele, genauer gesagt das Versenden der Nachrichten. Bei Internet-Multiplayer-Spielen muss eine Nachricht gleichzeitig an mehrere Empfänger gesendet werden. (Das Internet kann man sich dabei als ein Netz von sogenannten *Routern* vorstellen.) Man verbindet einen Rechner mit dem Internet, indem er an einen Router in diesem Netz angeschlossen wird. Für Multiplayer-Spiele ist nun wichtig, einen sogenannten *gemeinsamen Gruppenbaum*[10,], also ein Teil-Netz zu finden, das alle Router enthält, an denen die Spieler angeschlossen sind. Der gemeinsame Gruppenbaum verbindet also zur Spielergruppe gehörende Router. Meistens gewichtet man die Verbindung zwischen zwei Routern entsprechend ihrer Distanz, und sucht nach einem Gruppenbaum mit minimalen Kosten.

Obige Situation ist in der Abbildung 1.1 visualisiert. Hier ist ein kleines Netz und ein gemeinsamer Gruppenbaum zu sehen. Das Netz besteht aus den Router a, b, c, d, e, f und aus einer Gruppe von drei Spielern, die an die Router a, b, f angeschlossen sind. Der Gruppenbaum enthält die Verbindungen ac, bc, ce, ef und kostet 13(=3+3+3+4).

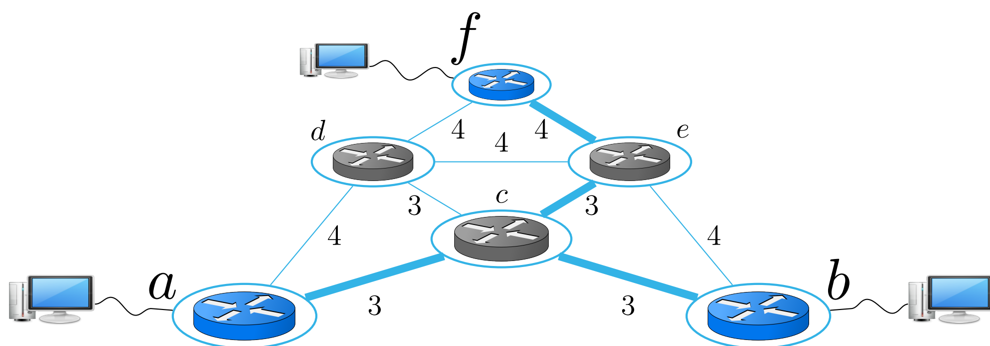


Abbildung 1.1: Netz und Gruppenbaum.

Das Problem, einen gemeinsamen Gruppenbaum zu berechnen, ist in der Graphentheorie als *das Steinerbaumproblem(**STP**)* bekannt und ist NP-schwer. In dieser Arbeit wird das Problem auf der Basis eines *dynamischen Programmieransatzes* untersucht. Die dynamische Programmierung kann benutzt werden, wenn das Problem sich in kleine Teilprobleme zerlegen lässt. Als Zerlegungstechnik werden wir die sogenannte *Baumzerlegung* verwenden. Die Baumzerlegung zeigt die Ähnlichkeit eines Graphen zu einem Baum. Ein Maß für die Qualität der Baumzerlegung ist die sogenannte *Baumweite*. Je kleiner die Baumweite einer Baumzerlegung ist, desto besser ist die Zerlegung und desto kleiner werden die Teilprobleme. Die Baumzerlegung mit kleinster Baumweite zu finden ist jedoch NP-schwer [4]. Für die Graphen $G = (V, E)$ mit Baumweite tw gibt es schon einige baumzerlegungsbasierte Algorithmen, die das Steinerbaumproblem lösen:

- Algorithmus [9] mit Laufzeit $O(\boldsymbol{tw^{4tw}} \cdot |V|)$;
- Algorithmus [5] mit Laufzeit $O(\boldsymbol{B^2_{tw+2}} \cdot tw \cdot |V|)$, wobei $B_k < k!$ die Anzahl der Partitionen einer k-elementigen Menge ist;
- Algorithmus [18] mit Laufzeit $O(h \cdot \boldsymbol{(2tw)^{|T|}})$, wobei $T \subseteq V$ die Menge der Terminale und $h = O(|V|)$ die Höhe der Baumzerlegung ist.

Die Faktoren, die in der jeweiligen Laufzeit dieser Algorithmen dominieren, sind fett geschrieben. Hier wird eine Heuristik für das Steinerbaumproblem mit kleinerem Faktor, nämlich mit $\boldsymbol{2^{tw}}$ vorgestellt. Die Laufzeit des Algorithmus beträgt $F + O\left(|V| \cdot \boldsymbol{2^{tw}} \cdot tw^3\right)$, wobei F eine Zufallsvariable mit Erwartungswert $O\left(|V| \cdot 2^{tw} \cdot tw\right)$ ist. Dabei ist die Wahrscheinlichkeit, dass F größer als $|V| \cdot 2^{tw} \cdot tw^3$ ist, ist kleiner als $2^{-|V| \cdot tw^2}$.

1.2 Aufbau der Arbeit

In Kapitel 2 werden die grundlegende Datenstrukturen behandelt und einige Notationen vorgestellt, die in dieser Arbeit benutzt werden. Der Algorithmus für das Steinerbaumproblem ist in Kapitel 3 beschrieben. In Unterkapitel 3.1 wird die zentrale Beobachtung des Algorithmus vorgestellt. Da der Algorithmus ein dynamisches Programm ist, müssen die Teillösungen effizient verwaltet werden. Das wird in Unterkapitel 3.2 behandelt. Der Algorithmus für das Steinerbaumproblem in allgemeinen Graphen enthält viele Details. Daher stellen wir zunächst die Kernidee des Algorithmus in Unterkapitel 3.3 anhand eines Spezialfalls vor. Damit diese Idee auch auf allgemeinen Graphen anwendbar ist, muss ein Problem beseitigt werden. Dazu wird in Unterkapitel 3.4 ein Trick vorgestellt. Anschließend wird in Unterkapitel 3.5 der Algorithmus für das Steinerbaumproblem in allgemeinen Graphen beschrieben. Die Analyse des Algorithmus wird in Unterkapitel 3.6 durchgeführt. Der Algorithmus wurde implementiert und auf bekannten Benchmark-Instanzen getestet. Das werden wir in Kapitel 4 behandeln. Mit der Zusammenfassung und dem Ausblick (Kapitel 5) werden wir die Arbeit abschließen.

Kapitel 2

Grundlagen

Viele Probleme wie das *gemeinsame Gruppenbaumproblem* aus der Einführung lassen sich mit Hilfe der Graphentheorie gut beschreiben. In diesem Kapitel werden einige grundlegende Datenstrukturen behandelt. Die Definitionen und Lemmata in den Unterkapiteln 2.1 und 2.3 sind aus dem Buch [6] und aus den Vorlesungsskripten [17], [3,] und [15] leicht modifiziert übernommen.

2.1 Graph

Ein (ungerichteter) *Graph* $G = (V, E)$ besteht aus der Knotenmenge V und der Kantenmenge $E \subseteq \{\{u, v\} : u, v \in V\}$. Eine Kante u, v verbindet die Knoten u und v. Beispielsweise kann das Router-Netz aus der Abbildung 1.1 wie folgt als ein Graph modelliert werden: Die Router entsprechen den Knoten und die Verbindungen zwischen je zwei Routern den Kanten (siehe Abbildung 2.1). Ein gerichteter *Graph* $G = (V, E)$ besteht aus der Knotenmenge V und der Kantenmenge E, wobei jede Kante $(u, v) \in E$ ein geordnetes Knotenpaar ist. Mit $V(G)$ beziehungsweise $E(G)$ bezeichnen wir die Knotenmenge beziehungsweise die Kantenmenge eines Graphen G.

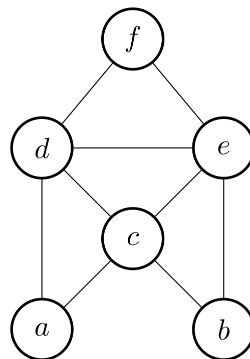


Abbildung 2.1: Ein Graph

Zwei Knoten $u, v \in V$ heißen *adjazent*, oder *benachbart*, wenn sie durch eine Kante verbunden sind. Man kann einen Graphen als *Adjazenzlisten* darstellen, indem für jeden Knoten v die Liste seiner Nachbarn (*Adjazenzliste*) gespeichert wird. Die Adjazenzlisten-Darstellung braucht $O(|V| + |E|)$ Speicherplatz. Ein Knoten $v \in V$ und eine Kante $e \in E$ heißen *inzident*, wenn $v \in e$ ist. Der Graph heißt *vollständig*, wenn je zwei Knoten $u \neq v \in V$ benachbart sind. Ein *Teilgraph* H von G ist ein Graph, so dass $V(H) \subseteq V(G)$ und $E(H) \subseteq E(G)$ ist. Für $B \subseteq V$ bezeichnen wir mit $G[V \setminus B]$ den Teilgraphen von G, den man erhält, indem man die Knoten B mit allen inzidenten Kanten aus G entfernt. Diesen Teilgraphen nennt man auch den von $V \setminus B$ *induzierten Teilgraph*. Für die Teilgraphen H_1, H_2 von G bezeichnen wir mit $H_1 \cap H_2$ einen Teilgraphen von G mit der Knotenmenge $V(H_1) \cap V(H_2)$ und der Kantenmenge $E(H_1) \cap E(H_2)$. Analog dazu ist $H_1 \cup H_2$ definiert. Ein *Pfad* P in Graph G ist eine Sequenz $v_0, e_1, v_1, \cdots, e_k, v_k$ der Knoten $v_0, \cdots, v_k$ und Kanten $e_i = \{v_{i-1}, v_i\}$ für jedes $1 \leq i \leq k$ mit $k \geq 1$. Man bezeichnet P als (v_0, v_k)-Pfad. P heißt *einfach*, wenn $v_0, \cdots, v_k$ unterschiedlich sind. P heißt *Kreis*, wenn nur v_0 und v_k identisch sind. Der Graph heißt *zusammenhängend*, wenn er für jedes Knotenpaar $v, w \in V$ einen (v, w)-Pfad enthält. Der Graph heißt *Wald*, falls er keinen Kreis enthält. Ein zusammenhängender Wald heißt *Baum*.

Ein gewurzelter Baum $J = (V, E)$ ist ein gerichteter Baum mit einer Wurzel $r \in V$. Sei (u, v) eine Kante von J. Wir bezeichnen u als *Elter* von v und v als Kind von u. Nur die Wurzel hat keinen Elter. Jeder Knoten außer der Wurzel hat genau einen Elter. Ein Knoten, der kein Kind hat, heißt *Blatt*.

Für jeden Knoten v bezeichnen wir mit V_v die Menge der Knoten w, so dass der Graph einen (v, w)-Pfad enthält. Der induzierte Teilgraph $G(v) := G[V_v]$ heißt *Zusammenhangskomponente* oder *Komponente* von G.

Mit $G = (V, E, c)$ bezeichnen wir einen *gewichteten* Graphen, in dem die Kante e das Gewicht bzw. die Kosten c_e hat. Die *Kosten eines Teilgraphen H* werden definiert durch die Summe der Kosten seiner Kanten. Sei G zusammenhängend. Ein *Spannbaum H* auf G ist ein Teilgraph von G, der alle Knoten von G enthält und ein Baum ist. Sei G zusammenhängend und $c : E \rightarrow \mathbb{R}$. Der *minimale Spannbaum (eng. Minimum Spanning Tree, kurz MST)* ist ein Spannbaum mit minimalen Kosten.

2.2 Steinerbaumproblem

Nun können wir das *gemeinsame Gruppenbaumproblem* als ein Graphproblem formulieren. Der ungerichteter Graph $G = (V, E)$ enthält für jeden Router v einen *Knoten* $v \in V$, für jede Verbindung zwischen zwei Routern v, w eine *Kante* $\{v, w\} \in E$, wobei die Kante entsprechend der Verbindung gewichtet ist. Die auserwählte Router, an denen die Spieler angeschlossen sind, bezeichnen wir als *Terminale* $T \subseteq V$. Der gemeinsame Gruppenbaum entspricht dann dem folgenden Problem:

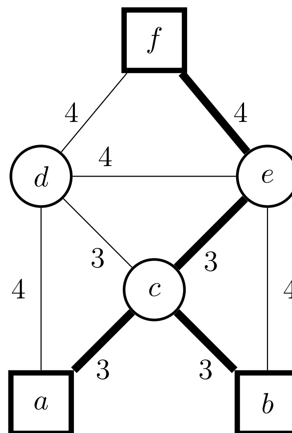


Abbildung 2.2: Ein Steinerbaum (fette Kanten)

Steinerbaumproblem

Gegeben ist ein (ungerichteter) Graph $G = (V, E)$, die Kostenfunktion $c : E \to \mathbb{R}^+$ und Terminale $T \subseteq V$. Ein Steinerbaum in G ist ein zusammenhängender Teilgraph von G, der alle Terminale enthält. Im Steinerbaumproblem wird ein Steinerbaum $H = (V_H, E_H)$ gesucht, so dass $\sum(c_e : e \in E_H)$ minimal ist.

In Abbildung 2.2 ist ein Steinerbaum mit fetten Kanten hervorgehoben, der dem gemeinsamen Gruppenbaum aus Abbildung 1.1 entspricht. Die rechteckigen Knoten sind die Terminale.

2.3 Baumzerlegung

Viele für allgemeine Graphen schwierige Probleme können auf Bäumen effizient gelöst werden. Man fragt sich daher, wie baumähnlich ein Graph ist. Wenn wir uns beispielsweise den Graph $G = (V, E)$ aus der Abbildung 2.1 näher ansehen, erkennen wir die folgende Ähnlichkeit zu einem Baum J. Man könnte sich die Knoten $\{a, c, d\} \subseteq V$ als einen Baumknoten $A \in V(J)$, $\{b, c, e\} \subseteq V$ als $B \in V(J)$, $\{c, d, e\} \subseteq V$ als $C \in V(J)$ und $\{d, e, f\} \subseteq V$ als $D \in V(J)$ vorstellen. Außerdem besteht $E(J)$ aus den Kanten $\{AC, BC, CD\}$. Diese Überlegung ist in der Abbildung 2.3 dargestellt.

Formal kann man die Baumzerlegung wie folgt definieren. Eine *Baumzerlegung* von $G = (V, E)$ ist ein Paar $(J, \mathcal{X})$ mit einem Baum J und einer Familie $\mathcal{X} = \{X_i : i \in V(J)\}$ von Teilmengen $X_i \subseteq V$, so dass:

tw1: Für jeden Knoten $v \in V$ gibt es einen Baumknoten $i \in V(J)$ mit $v \in X_i$.

tw2: Für jede Kante $e \in E$ gibt es einen Baumknoten $i \in V(J)$ mit $e \in X_i$.

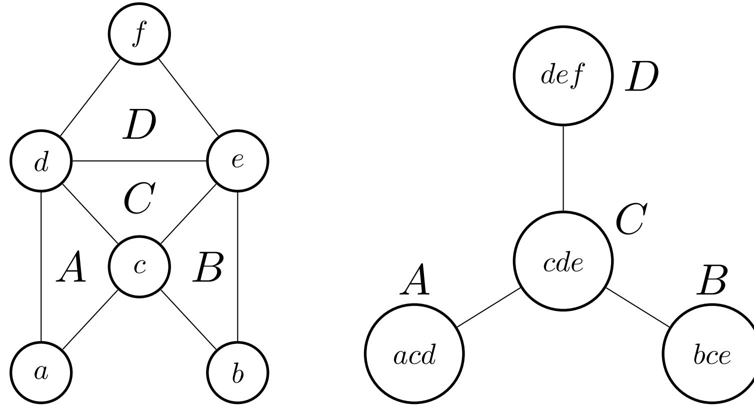


Abbildung 2.3: Ein Graph (links) und seine Baumzerlegung (rechts)

tw3: Für jeden Knoten $v \in V$ ist der induzierte Teilgraph $J[\{i : v \in X_i\}]$ zusammenhängend.

Eine Knotenmenge $X \in \mathcal{X}$ wird oft als *Bag* bezeichnet. Für einen Graphen kann man sehr leicht eine Baumzerlegung $(J, \mathcal{X})$ berechnen. Beispielsweise kann man den ganzen Graph in ein Bag einpacken: $\mathcal{X} = \{V\}$. Doch damit haben wir nichts gewonnen. Unser Ziel ist allerdings häufig den Graphen in möglichst kleine Teile zu zerteilen, damit wir an kleinen Teilen arbeiten. Ein Maß für die Qualität der Baumzerlegung ist die Baumweite. Die *Baumweite tw einer Baumzerlegung* $(J, \mathcal{X})$ ist die Größe des größten Bags minus eins:

$$tw(J, \mathcal{X}) := \max\{|X| : X \in \mathcal{X}\} - 1.$$

Die *Baumweite* $tw(G)$ *eines Graphen* G ist die kleinste Baumweite über alle Baumzerlegungen von G.

Je kleiner also die Baumweite einer Baumzerlegung ist, desto besser ist die Zerlegung. Die beste Baumzerlegung hat dann die kleinste Baumweite, nämlich $tw(G)$. Diese Zerlegung zu finden ist jedoch NP-schwer [4].

Wir betrachten jetzt einige Eigenschaften der Baumzerlegung. Bezüglich der Größe der Baumzerlegung werden wir Lemma 6.3 aus [15] ohne den Beweis wiedergeben:

2.3.1 Lemma. *Ist* $(J, \mathcal{X})$ *eine Baumzerlegung des Graphen* G *mit* $X_i \setminus X_j \neq \emptyset$ *für jede Kante* $\{i, j\} \in E(J)$, *so gilt:*

$$|V(J)| \leq |V(G)|.$$

Angenommen, wir haben eine Baumzerlegung $(J, \mathcal{X})$ für den Graph G. Wie können wir G mit Hilfe von $(J, \mathcal{X})$ zerlegen? Als Zerlegungsbegriff benutzt man Separatoren. Ein *Separator* im Graphen $G = (V, E)$ ist eine Teilmenge von V, nach deren Entfernung aus G dieser in mehrere Zusammenhangskomponenten zerfällt. Auch hier werden wir Lemma 6.4 aus [15] ohne Beweis wiedergeben:

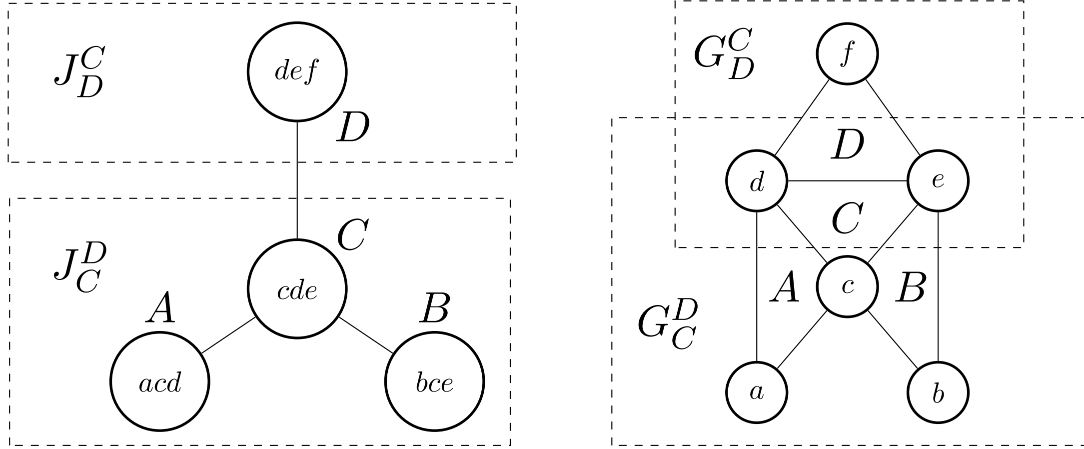


Abbildung 2.4: Teilbäume J_C^D, J_D^C und Teilgraphen G_C^D, G_D^C

2.3.2 Lemma. *Ist $(J, \mathcal{X})$ eine Baumzerlegung des Graphen G mit $X_i \setminus X_j \neq \emptyset$ für jede Kante $\{i, j\} \in E(J)$, so gilt:*

Für jede Kante $\{i, j\} \in E(J)$ ist $X_i \cap X_j$ ein Separator in G.

Abschließend werden wir Satz 2.27 aus [3] ohne den Beweis wiedergeben:

2.3.3 Lemma. *Für alle Graphen $G = (V, E)$ gilt $|E| \leq tw(G) \cdot |V|$.*

2.4 Notationen

Um den Algorithmus besser zu beschreiben, brauchen wir noch einige Notationen:

***Knoten* $\leftrightarrow$ *Baumknoten*:** Wir werden die Knoten des Graphen mit *Knoten* und die des Baums mit *Baumknoten* bezeichnen, um sie besser zu unterscheiden.

***Kanten*:** Die Kante $\{i, j\}$ wird kurz mit ij bezeichnet.

$c(H) = \sum(c_e \colon e \in E(H))$: die Kosten eines Teilgraphen H von $G = (V, E, c)$;

***Baumknoten* $\leftrightarrow$ *Bag*:** Wegen der Übersichtlichkeit werde ich sowohl den Baumknoten $i \in V(J)$ als auch das dazugehörende Bag $X_i \in \mathcal{X}$ einfach mit X bezeichnen. X ist beispielsweise beim Schreiben *"$XY \in E(J)$"* der Baumknoten der Baumkante XY, dagegen bei *"$X \cap Y$"* als ein Bag zu verstehen.

J_X^Y, V_X^Y, G_X^Y: Entfernt man eine Kante XY aus dem Baum J, so zerfällt dieser in zwei Teilbäume. Sei J_X^Y der Teilbaum, der den Baumknoten X aber nicht Y enthält und V_X^Y die Vereinigung aller Bags in J_X^Y. Anstatt $G[V_X^Y]$ werden wir kurz G_X^Y schreiben.

Beispielsweise ist in Abbildung 2.4 der Teilbaum J_C^D der Baumzerlegung aus Abbildung 2.3 dargestellt. Der Teilbaum J_C^D besteht aus zwei Baumkanten AC, BC und

drei Baumknoten A, B, C. Die Knoten, die in Bags A, B, C enthalten sind, bilden V_C^D, also $V_C^D = \{a, b, c, d, e\}$. Der induzierte Teilgraph $G_C^D := G[V_C^D]$ ist auch in der Abbildung 2.4 zu sehen.

Kapitel 3

Algorithmus

3.1 Vorüberlegungen

Gegeben sei eine Steinerbaumproblem-Instanz durch $G = (V, E, c)$ und $T \subseteq V$. Sei $(J, \mathcal{X})$ eine Baumzerlegung mit Baumweite tw dieses Graphen, wobei $X \setminus Y \neq \emptyset$ für jede Kante $XY \in E(J)$ ist. Bezüglich einer Kante $XY \in E(J)$ teilen wir den Graph in zwei (überlappende) Teilgraphen, nämlich G_X^Y und G_Y^X. Nach Lemma 2.3.2 ist $X \cap Y$ ein Separator in G, also der überlappende Teil der beiden Teilgraphen ist $G[X \cap Y]$. Angenommen, die beiden Teilgraphen enthalten Terminale. Ein Steinerbaum H muss jedes Terminal-Paar (t_1, t_2) mit einem Pfad verbinden. Ein (t_1, t_2)-Pfad mit $t_1 \in G_X^Y$ und $t_2 \in G_Y^X$ ist nur über den überlappenden Teil $G[X \cap Y]$ möglich(siehe Abbildung 3.1). Das heißt, der Steinerbaum H induziert in G_X^Y einen Teilgraphen H_X^Y, der die folgende Bedingung erfüllt:

> ***H**** *: Jede Komponente von H_X^Y hat einen Knoten aus dem Separator $X \cap Y$. Außerdem enthält er alle Terminalknoten aus G_X^Y.*

Diese Beobachtung führt zu der folgenden Idee.

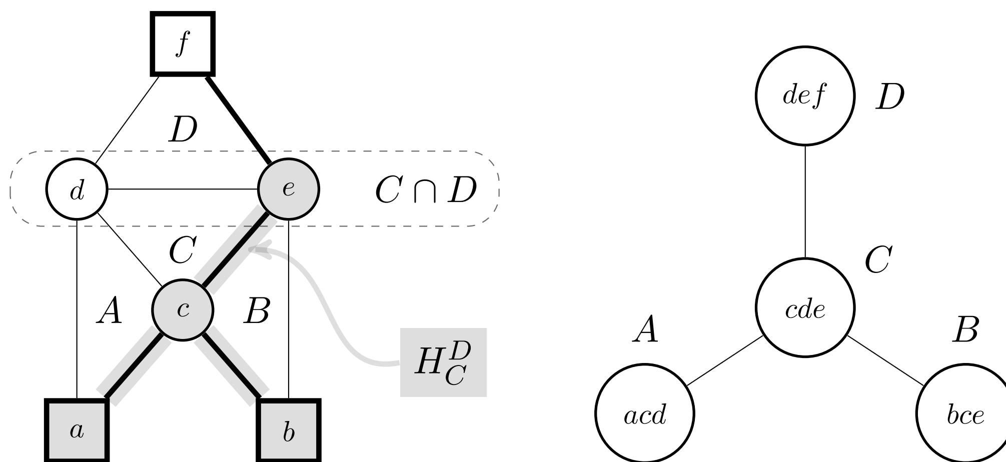


Abbildung 3.1: Separator $C \cap D$ und Teilgraph H_C^D

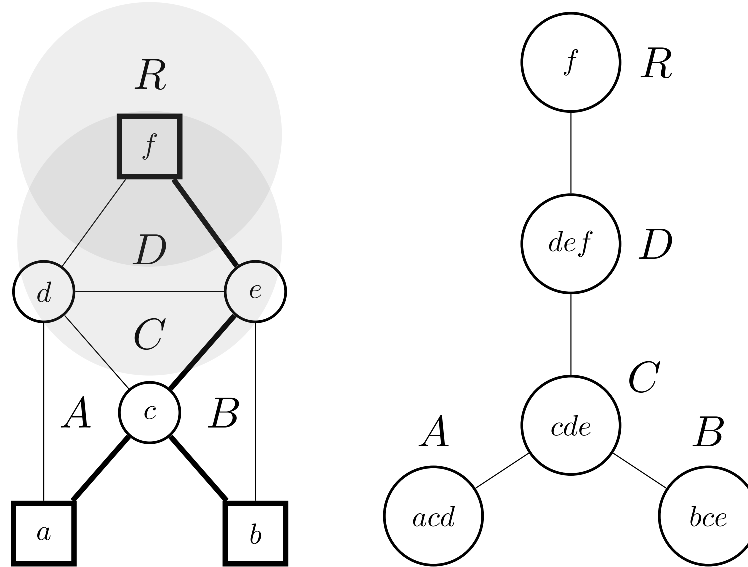


Abbildung 3.2: Baumzerlegung mit künstlicher Kante DR

3.1.1 Idee. Sei W ein Bag, der mindestens einen Terminalknoten t enthält. Füge künstlich eine Kante WR mit $R = \{t\}$ in J (siehe Abbildung 3.2). Wähle R zur Wurzel von J. Wenn wir nun für jeden Baumknoten $X \neq R$ mit Elter Y einen Teilgraphen H_X^Y von G_X^Y so konstruieren könnten, dass **H*** erfüllt ist und die Summe $\sum(c_e : e \in E(H_X^Y))$ minimal ist, dann ist H_W^R nichts anders als der optimale Steinerbaum. Denn $W \cap R = \{t\}$ besteht nur aus einem Knoten t. Wegen **H*** enthalten alle Komponente von H_W^R den Knoten t. D.h. H_W^R besteht nur aus einer einzigen Komponente und ist daher zusammenhängend. Laut **H*** enthält H_W^R jeden Terminalknoten aus $G_W^R = G$, also alle Terminalknoten. Somit ist H_W^R ein zusammenhängender Teilgraph von G, der alle Terminale T enthält, so dass $\sum(c_e : e \in E(H_X^Y))$ minimal ist. Dann ist er ein optimaler Steinerbaum.

3.1.2 Bemerkung. (Separatoren) Bezeichnen wir J vor der Modifikation mit J_{old}. Da wir den Graphen nicht modifiziert haben, ist $X \cap Y$ für jede Kante $XY \in E(J_{old})$ nach Lemma 2.3.2 ein Separator. Da jede Kante $XY \neq RW$ von J auch in J_{old} ist, ist $X \cap Y$ weiterhin ein Separator. Dagegen ist $W \cap R$ kein Separator, da $R \backslash W$ leer ist und somit die künstliche Kante RW die Voraussetzung des Lemmas 2.3.2 nicht erfüllt. Das ist aber kein Problem. Ob $W \cap R$ ein Separator ist, ist nur deswegen interessant, um festzustellen ob H_W^R einen Knoten aus $W \cap R$ enthalten soll oder nicht. Aber $W \cap R$ besteht nach Konstruktion nur aus einem einzigen Terminalknoten und ein Terminalknoten muss unbedingt ausgewählt werden.

3.1.3 Bemerkung. Bevor wir den Baum J durch eine künstliche Kante erweitert hatten, hatte J wegen Lemma 2.3.1 höchstens $|V|$ Knoten. Durch die Modifikation wurde J nur um einen Baumknoten erweitert. Daher hat J nach der Modifikation höchstens $|V| + 1$ Baumknoten.

Mit Hilfe der dynamischen Programmierung wollen wir der Idee 3.1.1 nachgehen. Wir starten bei den Blättern von J und konstruieren Bottom-Up für jeden Baumknoten $X \neq R$ mit Elter Y den Teilgraphen H_X^Y. Irgendwann berechnen wir auch H_W^R und sind fertig. Wie können wir aber H_X^Y möglichst billig konstruieren?

3.1.4 Lemma. *(MST Eigenschaft) Betrachte einen Teilbaum H des Steinerbaums, wobei H nur die Knoten $v_1, \cdots, v_j$ enthält. Dann ist H ein MST auf $G[v_1, \cdots, v_j]$.*

Beweis. Da H ein Teilbaum des Steinerbaums ist, ist er zusammenhängend und daher ein Spannbaum auf $G[v_1, \cdots, v_j]$. Falls H kein MST ist, kann man es durch so einen ersetzen. Der dadurch geänderte Steinerbaum enthält weiterhin dieselbe Knoten ist aber billiger. Widerspruch zur Optimalität des Steinerbaums! □

Nach Lemma 3.1.4 kann das Steinerbaumproblem in zwei Teilaufgaben zerlegt werden. Sei H der Steinerbaum.

1. Finde die Knotenmenge $V(H)$ des Steinerbaums.

2. Berechne MST auf $G[V(H)]$.

Wenn man also H_X^Y im Teilgraphen $G[S]$ für jede Teilmenge $S \subseteq V_X^Y$ konstruieren könnte, würde man für $V_X^Y = V$ auch die Knotenmenge des optimalen Steinerbaums, also $S = V(H)$ betrachten. Die Sache hat aber einen Hacken: $V_X^Y = V$ hat $2^{|V|}$ Teilmengen. Sie können wir nicht einmal aufschreiben.

3.1.5 Idee. Diese Beobachtung führt uns auf eine Idee:

a) Für jede Teilmenge $S \subseteq X$ versuche H_X^Y so zu konstruieren, dass er aus X nur die Knotenmenge S enthält, also $V(H_X^Y) \cap X = S$. Leider darf nicht jede Teilmenge $S \subseteq X$ benutzt werden. Das werden wir in Abschnitt 3.1.1 ausführlich diskutieren.

b) Jede Komponente $K \in H_X^Y$ ist ein MST auf $G[V(K)]$.

3.1.1 Zulässige Knotenteilmenge

Man beachte, dass die Bedingung **H*** nicht für jede Teilmenge S eines Bags X erfüllbar ist. Beispielsweise muss jeder Terminalknoten aus X auch in S sein:

$$S \cap T = X \cap T \tag{3.1}$$

Die Gleichung (3.1) sorgt dafür, dass alle Terminalknoten gewählt werden. Diese Gleichung reicht alleine nicht aus. Die Terminalknoten müssen nicht nur gewählt werden, sondern auch zusammengehängt werden. Wir erinnern uns an die Überlegungen am Anfang dieses Kapitels. Wenn es sowohl in V_Y^X als auch in V_X^Y Terminalknoten gibt, dann muss

mindestens ein Knoten aus dem Separator $X \cap Y$ gewählt werden. Nehmen wir an, V_Y^X hat einen Terminalknoten t und S ist eine nichtleere Teilmenge von X, so dass S keinen Knoten aus dem Separator $X \cap Y$ hat. Dann gibt es keine Verbindung zwischen einem Knoten $v \in S$ und t. Eine Teilmenge $\emptyset \neq S \subseteq X$, die keinen Knoten aus dem Separator $X \cap Y$ enthält, kann daher nicht zu einer Lösung mit endlichen Kosten führen. Somit können nur die Teilmengen $S \subseteq X$ mit folgender Eigenschaft zu einer Lösung mit endlichen Kosten führen:

$$V_Y^X \cap T \neq \emptyset \Longrightarrow S = \emptyset \text{ oder } S \cap (X \cap Y) \neq \emptyset. \tag{3.2}$$

Aus Symmetriegründen gilt auch für jede Teilmenge $P \subseteq Y$ Folgendes:

$$V_X^Y \cap T \neq \emptyset \Longrightarrow P = \emptyset \text{ oder } P \cap (X \cap Y) \neq \emptyset. \tag{3.3}$$

In Bedingung (3.2) ist die linke Seite wegen der Konstruktion immer erfüllt. Denn wegen der Idee 3.1.1 hat die Wurzel R, somit V_Y^X immer einen Terminalknoten. Daher können wir diese Bedingung wie folgt verkürzen:

$$S = \emptyset \text{ oder } S \cap (X \cap Y) \neq \emptyset. \tag{3.4}$$

Die Bedingung (3.3) kann auch verkürzt werden. Da V_Y^X wegen der Idee 3.1.1 immer einen Terminalknoten hat, muss im Falle $V_X^Y \cap T \neq \emptyset$ mindestens ein Knoten aus dem Separator $X \cap Y$ gewählt werden. Daher muss P mindestens einen Knoten aus diesem Separator enthalten. Daraus folgt:

$$V_X^Y \cap T \neq \emptyset \Longrightarrow P \cap (X \cap Y) \neq \emptyset. \tag{3.5}$$

Wir fassen zusammen: Während die Bedingung (3.1) dafür sorgt, dass alle Terminalknoten gewählt werden, sorgen die Bedingungen (3.4) und (3.5) für deren Zusammenhang. Wir müssen noch feststellen, wieviel Zeit die Überprüfung dieser Bedingungen in Anspruch nimmt.

Wir erstellen einmalig ein Array $isTerminal$ der Länge $|V|$, so dass für jeden Knoten $v \in V$ gilt: $isTerminal[v] = 1$, falls $v \in T$ ist, sonst $isTerminal[v] = 0$. Die Konstruktion kostet einmalig $|V|$ Zeit. Danach kann zu jeder Zeit in $O(1)$ Zeit abgefragt werden, ob ein Knoten Terminalknoten ist.

Mit dem obigen Trick kann man für ein Bag X den Durchschnitt $X \cap T$ in $O(tw)$ Zeit berechnen, da ein Bag höchstens $tw + 1$ Knoten hat. Das gilt natürlich auch für jede Teilmenge $S \subseteq X$. Somit kann für einen Baumknoten X und eine Teilmenge $S \subseteq X$ die Bedingung (3.1) in $O(tw)$ Zeit überprüft werden.

Für einen Baumknoten X mit Elter Y und für Teilmenge $S \subseteq X$ kann die Bedingung (3.4) in $O(tw^2)$ Zeit berechnet werden. Der Grund dafür ist Folgendes. Ob S leer ist, kann man in konstanter Zeit feststellen. Wegen $S \subseteq X$ gilt $S \cap (X \cap Y) = S \cap Y$. Man kann

jeden Knoten aus S mit jedem Knoten in Y vergleichen. Da S und Y jeweils höchstens $tw + 1$ Knoten haben, braucht man wie behauptet $O(tw^2)$ Zeit.

Analog zur Bedingung (3.4) kann man zeigen, dass die Überprüfung der rechten Seite der Bedingung (3.5) auch $O(tw^2)$ Zeit braucht. Wie lange dauert die Überprüfung von $V_X^Y \cap T \neq \emptyset$?

Überprüfung von $V_X^Y \cap T \neq \emptyset$. Um die Beschreibung eindeutig zu machen, unterscheiden wir ausnahmsweise an dieser Stelle den Baumknoten $x \in V(J)$ vom dazugehörenden Bag $X_x \in \mathcal{X}$. Wir überprüfen die Bedingung $V_x^y \cap T \neq \emptyset$. Sei r die Wurzel des Baums J und $x \neq r$ ein beliebiger Baumknoten mit Elter y und Kindern $w_1, \cdots, w_l$. Es gilt:

$$V_x^y = X_x \cup \left(\bigcup_{w \in \{w_1, \dots, w_l\}} V_w^x \right).$$

Dann gilt:

$$V_x^y \cap T = (X_x \cap T) \cup \left(\bigcup_{w \in \{w_1, \dots, w_l\}} (V_w^x \cap T) \right). \tag{3.6}$$

Wir definieren eine Funktion A, wobei $A(x) = 1$ für jeden Baumknoten $x \neq r$, falls $V_x^y \cap T \neq \emptyset$ ist, sonst $A(x) = 0$. Diese Funktion kann mit dynamischer Programmierung effizient berechnet werden. Aus der Gleichung (3.6) folgt:

$$A(x) = 1 \iff (X_x \cap T \neq \emptyset) \text{ oder } A(w) = 1 \text{ für ein Kind } w \text{ von } x. \tag{3.7}$$

Die Funktion A wird als ein Array realisiert. Initial setzen wir $A[x] := 0$ für jeden Baumknoten $x \neq r$. Wir starten bei den Blättern des Baums J und durchlaufen seine Baumknoten in folgender Reihenfolge: Ein Baumknoten x wird erst dann bearbeitet, nachdem alle seine Kinder bearbeitet wurden. Dabei aktualisieren wir den Wert von $A[x]$ wie folgt: Wir setzen $A[x] := 1$ falls gilt:

$$(X_x \cap T \neq \emptyset) \text{ oder } A[w] = 1 \text{ für ein Kind } w \text{ von } x. \tag{3.8}$$

3.1.6 Lemma. *Das obige dynamische Programm berechnet in $O(|V| \cdot tw)$ Zeit für jeden Baumknoten $x \neq r$ den Wert $A[x]$ gemäß der Aussage (3.7).*

Beweis. Zuerst zeigen wir die Korrektheit mit vollständiger Induktion.

IA: Sei $x \neq r$ ein Blatt. Initial ist $A[x] = 0$. Falls $X_x \cap T \neq \emptyset$ ist, wird $A[x] := 1$ gesetzt, sonst bleibt weiterhin $A[x] = 0$. Es gilt somit: $A[x] = 1 \iff (X_x \cap T \neq \emptyset)$. Da x kein Kind hat, erfüllt $A[x]$ die Aussage (3.7).

IS: Sei $x \neq r$ ein innerer Baumknoten. Wegen der Konstruktion sind alle Kinder von x bereits bearbeitet. Initial ist $A[x] = 0$. Der Algorithmus setzt $A[x] := 1$ falls (3.8) erfüllt ist. Wegen der Induktionsvoraussetzung ist $A[w]$ für jedes Kind w von x korrekt berechnet (es gilt also $A[w] = A(w)$), daher ist die rechte Seite der Aussage (3.7) erfüllt. Somit erfüllt auch $A[x]$ die Aussage (3.7).

Als Nächstes zeigen wir die Aussage über die Laufzeit. Nach Bemerkung 3.1.3 gibt es $O(|V|)$ Baumknoten. Für jeden Baumknoten $x \neq r$ wird $X_x \cap T$ berechnet. Wir haben oben gesehen, dass dies für einen Baumknoten $O(tw)$ Zeit kostet. Das kostet dann für alle Baumknoten insgesamt $O(|V| \cdot tw)$ Zeit. Außerdem muss für jeden Baumknoten x die Adjazenzliste $w_1, \cdots, w_l$ durchlaufen werden und die Werte $A[w_1], \cdots, A[w_l]$ jeweils in konstanter Zeit abgelesen werden. Dabei wird die Adjazenzliste eines Baumknotens x nur einmal und nur von x aus durchlaufen. Der Baum J hat $|V(J)| = O(|V|)$ Baumknoten, daher auch $|E(J)| = O(|V|)$ Baumkanten. Alle Adjazenzlisten können dann in $O(|V(J)| + |E(J)|) = O(|V|)$ Zeit durchlaufen werden. Die Laufzeit beträgt dann insgesamt $O(|V| \cdot tw + |V|)$ Zeit, was zu zeigen war. □

Wir halten die Ergebnisse dieses Abschnitts in einem Lemma fest.

3.1.7 Lemma. *Die effieziente Überprüfung der Bedingungen (3.1), (3.4) und (3.5) braucht einmalig $O(|V| \cdot tw + tw^2)$ Zeit. Danach kostet jede Überprüfung $O(tw^2)$ Zeit.*

Beweis. Die Konstruktion des Arrays $isTerminal$ kostet einmalig $O(|V|)$ Zeit. Danach kann die Bedingung (3.1) immer in $O(tw)$ Zeit überprüft werden. Die Bedingung (3.4) kann in $O(tw^2)$ Zeit überprüft werden. Nach Lemma 3.1.6 kann $A(x)$ einmalig für jeden Baumknoten $x \neq r$ in $O(|V| \cdot tw)$ Zeit berechnet werden. Danach können wir mit Hilfe $A(x)$ in konstanter Zeit abfragen, ob $V_x^y \cap T \neq \emptyset$ ist, das heißt, ob die linke Seite der Bedingung (3.5) erfüllt ist. Da die rechte Seite dieser Bedingung in $O(tw^2)$ Zeit berechnet werden kann, folgt die Behauptung. □

3.2 Teillösungen

Sei $(J, \mathcal{X})$ eine Baumzerlegung mit Baumweite tw des Graphen G. Wir werden für jedes Bag X der Baumzerlegung $(J, \mathcal{X})$ eine Tabelle tab_X erstellen. Sei der Baum J gemäß der Idee 3.1.1 mit den Baumknoten R gewurzelt. Sei $X \neq R$ mit Elter Y ein Baumknoten. Wegen Idee 3.1.5.a) soll tab_X für jede Teilmenge $S \subseteq X$ eine Zeile $tab_X(S)$ haben, die eine Teillösung, nämlich einen Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = S$ beschreibt.

Aus Effizienzgründen ist es nicht sinnvoll, die ganze Teillösung H_X^Y in einer tab_X-Zeile zu speichern. Denn H_X^Y liegt nicht unbedingt völlig in $G[X]$, sondern seine Teile sind eventuell in mehrere Teilgraphen $G[X]$, $G[Y], G[Z], \ldots$ für $X, Y, Z, \ldots \in \mathcal{X}$ gestreut. Stattdessen können wir seinen in $G[X]$ liegenden Teil, nämlich $H_X^Y \cap G[X]$ in tab_X, in $G[Y]$ liegenden

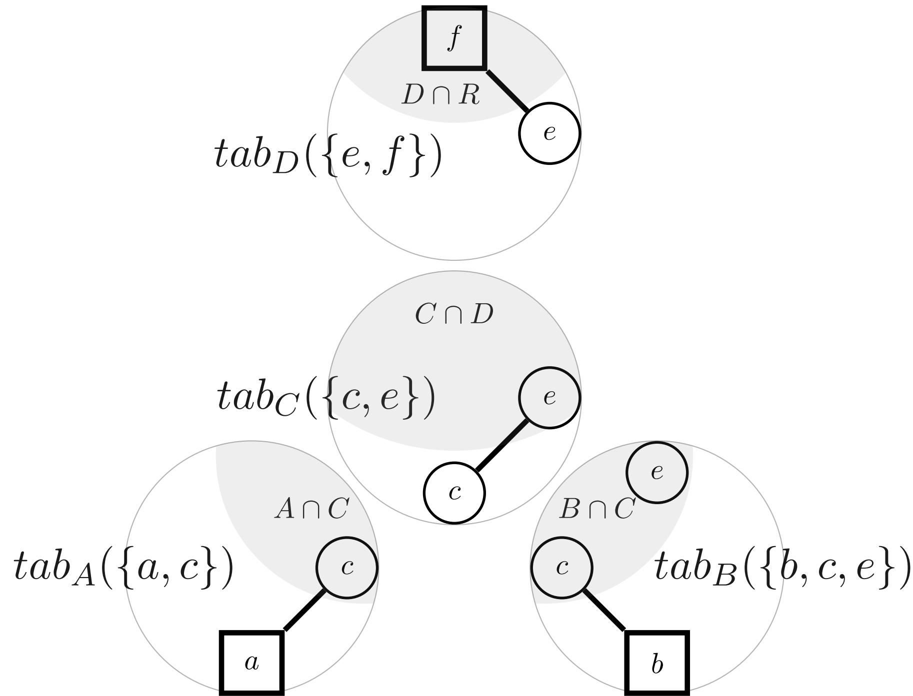


Abbildung 3.3: Die Teillösungen $tab_D(\{e, f\})$, $tab_C(\{c, e\})$, $tab_A(\{a, c\})$ und $tab_B(\{b, c, e\})$

$S \subseteq X$	Zeigerliste(Verkettung)	Komponente von $H_X^Y \cap G[X]$	$c(H_X^Y)$
...	...	...	
{c,e}	$tab_A(\{a, c\}), tab_B(\{b, c, e\})$	$(\{c, e\}, \{ce\})$	9
...	...	...	

Tabelle 3.1: Tabelle tab_X

Teil in tab_Y, usw. . . speichern und anschließend diese Teillösungen zusammenfügen. In Zeile $tab_X(S)$ werden wir dann die Teillösung $H_X^Y \cap G[X]$ speichern. Beispielsweise besteht der in Abbildung 3.2 dargestellte Steinerbaum $H = H_D^R$ aus den Teillösungen $tab_D(\{e, f\})$, $tab_C(\{c, e\})$, $tab_A(\{a, c\})$ und $tab_B(\{b, c, e\})$, die in Abbildung 3.3 zu sehen sind.

Die Tabelle tab_X kann beispielsweise wie die Tabelle 3.1 aussehen. Hier ist tab_C mit der Zeile $tab_C(\{c, e\})$ für Bag C aus Abbildung 3.2 dargestellt. Der Elter von C ist D, daher beschreibt diese Zeile den Teilgraphen H_C^D mit $V(H_C^D) \cap C = \{c, e\}$, der in die Teilgraphen $G[C]$, $G[A]$ und $G[B]$ gestreut ist. In $G[C]$ enthält H_C^D den Teilgraphen $H_C^D[\{c, e\}]$, der nur aus einem Baum mit zwei Knoten c, e und einer Kante ce besteht (siehe Abbildung 3.3). Um den Teilgraphen H_X^Y vollständig zu beschreiben, enthält die Zeile $tab_X(S)$ eine Zeigerliste. Beispielsweise zeigt der Eintrag $tab_A(\{a, c\})$ in dieser Liste auf eine Zeile in tab_A, nämlich auf die Zeile $tab_A(\{a, c\})$. Das liefert die folgende Information: Der Baumknoten C hat ein Kind A und der Teilgraph $H_C^D[\{c, e\}]$ ist mit dem Teilgraphen $H_A^C[\{a, c\}]$ aus der Tabelle tab_A verkettet. Das heißt, in $G[A]$ enthält H_C^D den Teilgraphen $H_A^C[\{a, c\}]$. Somit kann

man durch die rekursive Verfolgung der Zeiger der Zeile $tab_X(S)$ den Teilgraphen H_X^Y vollständig ablesen. Die Kosten $c(H_X^Y)$ speichern wir auch in einer Spalte ab (Tabelle 3.1).

Anmerkung: Die Teillösung H_X^Y mit $V(H_X^Y) \cap X = S$ für einen Baumknoten $X \neq R$ werden wir je nach Kontext mit H_X^Y oder mit $tab_X(S)$ bezeichnen. Manchmal ist es hilfreich zu wissen, um welchen Separator $X \cap Y$ es sich gerade handelt. Dann ist die Bezeichnung H_X^Y informativer. Manchmal aber ist es hilfreich zu wissen, welche Teilmenge $S \subseteq X$ in H_X^Y ist, dann ist $tab_X(S)$ besser.

3.3 Dynamisches Programm für einen Spezialfall

Der Algorithmus für allgemeine Graphen enthält viele kleine Details. Um seine wesentliche Ideen zu betrachten, sehen wir uns zuerst einen Algorithmus für folgenden einfachen Spezialfall an. Für den Graph G und seine Baumzerlegung $(J, \mathcal{X})$ mit Baumweite tw gilt:

Für jedes Bag $X \in \mathcal{X}$ ist $G[X]$ vollständig.

Sei der Baum J gemäß der Idee 3.1.1 mit R gewurzelt. Wir erstellen für jeden Baumknoten X eine Tabelle gemäß der Beschreibung im Unterkapitel 3.2. Die Zeile $tab_X(S)$ für jede $S \subseteq X$ wird dabei wie folgt initialisiert:

- Die Zeigerliste ist leer;
- jeder Knoten $v \in S$ ist eine Komponente $(\{v\}, \emptyset)$.
- Kosten $= 0$;

Sei Y der Elter von X. Die Zeile $tab_X(S)$ beschreibt dann die Teillösung H_X^Y, die am Anfang eventuell die Bedingung **H*** verletzt. Das dynamische Programm bearbeitet nun die Baumknoten von J eine nach dem anderen startend bei den Blättern hoch zur Wurzel und repariert quasi die verletzten Teillösungen. Genauer gesagt, dauert die Bearbeitung bis zum Kind W der Wurzel R. (Erinnerung: wegen der Idee 3.1.1 hat der Wurzel nur ein Kind.) Dabei wird die Bearbeitung eines Baumknotens erst dann angefangen, nachdem alle seine Kinder bearbeitet sind. Die Bearbeitung eines Baumknotens X mit Elter Y verläuft in zwei Phasen:

Phase 1: Konstruktion von H_X^Y mit $V(H_X^Y) \cap X = S$ für jede $S \subseteq X$.

Phase 2: Abgleich von tab_Y für tab_X (*Verkettung*).

Phase 1. Wir betrachten die Konstruktion eines Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = S$. Genauer gesagt, wir konstruieren sein in $G[X]$ liegenden Teil, nämlich $H_X^Y \cap G[X]$. Dieser Teilgraph H_X^Y wird in der Zeile $tab_X(S)$ verwaltet. Erfüllt S die Bedingungen (3.1) (jeder

Terminalknoten aus X muss in S sein) und (3.4) nicht, wird diese Zeile mit $+\infty$ bewertet und ihre Berechnung beendet. Angenommen, das Gegenteil wäre der Fall.

Falls H_X^Y eine Komponente hat, die keinen Knoten aus dem Separator $X \cap Y$ enthält, dann verletzt H_X^Y die Bedingung **H***. Das Ziel dieser Phase ist nun H_X^Y zu reparieren. Wir lassen seine Komponenten, die keinen Knoten aus dem Separator $X \cap Y$ enthalten, in $G[S]$ mit dem Algorithmus von Prim für MST (wegen der Idee 3.1.5.b)) solange wachsen, bis sie einen Knoten des Separators $X \cap Y$ enthalten. Sei $v \in S$ ein Knoten, so dass die Komponente $H_X^Y(v)$ keinen Knoten des Separators $X \cap Y$ hat. Starte Prim bei v in $G[S]$. Wegen der Voraussetzung ist $G[S]$ ein vollständiger Graph. Prim fügt in jeder Iteration eine neue Kante uw in diese Komponente ein. Wir inkrementieren die Kosten in Zeile $tab_X(S)$ um c_{uw} und brechen die Iteration ab, sobald $H_X^Y(v)$ mit einer Komponente $H_X^Y(v')$ mit $v' \in S$ zusammenhängt, die schon einen Knoten aus dem Separator $X \cap Y$ enthält.

Der Grund des obigen Abbruchs ist Folgendes: Die Phase 1 konstruiert für jede Knotenteilmenge $S \subseteq X$ *genau einen* Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = S$. Das führt aber zu einem Problem. In einer Baumzerlegung kann eine Knotenteilmenge $S \subseteq X$ in mehrere Bags $X' \neq X$ enthalten sein. Beispielsweise sei S sowohl in Y als auch bei seinen beiden Kindern X und X' enthalten. Der Algorithmus bearbeitet zuerst X und konstruiert genau einen Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = S$. Als Nächstes wird X' bearbeitet und genau ein Teilgraph $H_{X'}^Y$ mit $V(H_{X'}^Y) \cap X = S$ konstruiert. Es kann nun sein, dass durch die Kombination dieser Teillösungen Kreise in $H[S]$ entstehen. Da wir keinen weiteren Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = S$ konstruiert haben, können wir ihn nicht tauschen. Das gilt auch für $H_{X'}^Y$. Daher sorgen wir dafür, dass H_X^Y und $H_{X'}^Y$ keine Kante in $G[S]$ haben. Dazu haben wir obigen Abbruch gemacht.

Nur am Rande sei bemerkt, dass wegen dieses Abbruchs $H[S]$ und $H_X^Y[S]$ für $S \subseteq X \cap Y$ voneinander unterscheiden. $H_X^Y[S]$ ist zwar ein Teilgraph von $H[S]$, enthält aber keine Kante, während $H[S]$ Kanten haben kann. Betrachte dazu $S = B \cap C = \{c, e\}$ in Abbildung 3.3. Hier enthält $H[S]$ eine Kante ce, während $H_B^C[S]$ keine Kante hat.

3.3.1 Beispiel. Wir konstruieren H_X^Y mit $V(H_X^Y) \cap X = X$, wobei X, Y wie in der Abbildung 3.4 vorgegeben sind. Wir betrachten einen Ausschnitt der Tabelle tab_X. Die Zeile $tab_X(X)$ enthält initial den folgenden Eintrag:

$S \subseteq X$	$H_X^Y \cap G[X]$	$c(H_X^Y)$
...	...	...
$\{a, b, c, d, e\}$	$(\{a\}, \emptyset)$, $(\{b\}, \emptyset)$, $(\{c\}, \emptyset)$, $(\{d\}, \emptyset)$, $(\{e\}, \emptyset)$	0
...	...	...

Der Teilgraph $tab_X(X)$ hat fünf Komponenten, die jeweils aus einem Knoten bestehen. Die Komponenten $(\{a\}, \emptyset)$, $(\{b\}, \emptyset)$ und $(\{c\}, \emptyset)$ enthalten keinen Knoten aus dem Separator $X \cap Y$. Daher verletzt H_X^Y die Bedingung **H***. Wir lassen zuerst die Komponente $H_X^Y(a) = (\{a\}, \emptyset)$ wachsen (linkes Bild). Dazu wird Prim bei dem Knoten a gestartet und wird

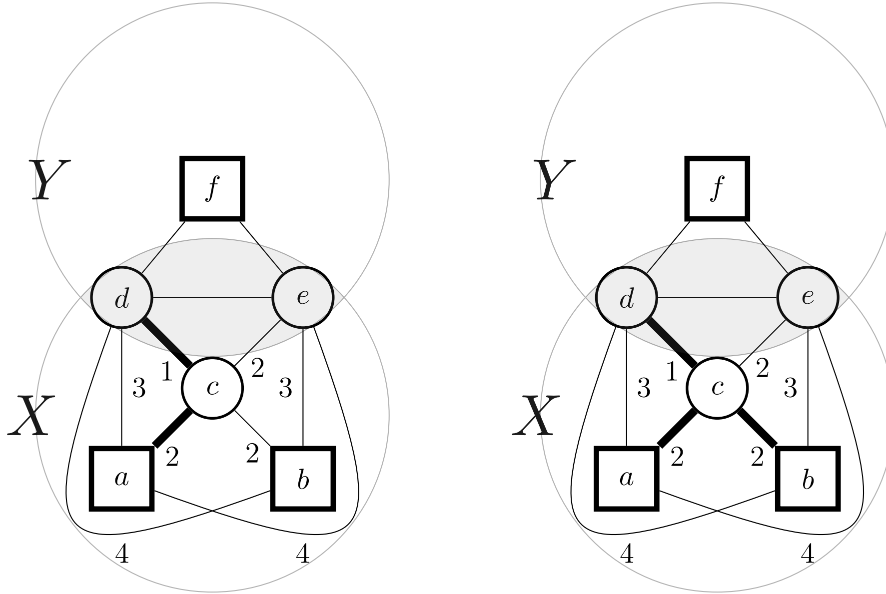


Abbildung 3.4: Konstruktion des Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = X$

gestoppt, sobald der Knoten d aus $X \cap Y$ erreicht wird. Nun enthält die Komponente $H_X^Y(a)$ die Knoten $\{a, c, d\}$ und der Teilgraph $H_X^Y \cap G[X]$ sieht wie folgt aus:

$S \subseteq X$	$H_X^Y \cap G[X]$	$c(H_X^Y)$
...	...	...
$\{a, b, c, d, e\}$	$(\{a, c, d\}, \{ac, cd\})$, $(\{b\}, \emptyset)$, $(\{e\}, \emptyset)$	3(=2+1)
...	...	...

Als Nächstes lassen wir die Komponente $H_X^Y(b) = (\{b\}, \emptyset)$ wachsen (Abb. 3.4 rechtes Bild). Dazu startet man Prim bei dem Knoten b. Von b aus wird direkt ein Knoten c aus der Komponente $H_X^Y(a)$ erreicht. Somit hängt $H_X^Y(b)$ mit $H_X^Y(a)$ und daher auch mit einem $X \cap Y$ Knoten d zusammen. Die Zeile $tab_X(X)$ wird wie folgt aktualisiert:

$S \subseteq X$	$H_X^Y \cap G[X]$	$c(H_X^Y)$
...	...	...
$\{a, b, c, d, e\}$	$(\{a, b, c, d\}, \{ac, bc, cd\})$, $(\{e\}, \emptyset)$	5(=3+2)
...	...	...

Nun enthält jede Komponente des Teilgraphen $H_X^Y \cap G[X]$ einen Knoten aus $X \cap Y$. Die Konstruktion von H_X^Y mit $V(H_X^Y) \cap X = X$ ist somit abgeschlossen.

Phase 2. Stellen wir uns vor, dass ein Teilgraph von H in $G[Y]$ durch $tab_Y(P)$ beschrieben ist. Die Frage ist nun, für welche $S \subseteq X$ soll $tab_Y(P)$ und $tab_X(S)$ verkettet werden. Auch hier müssen die Teilmengen $P \subseteq Y$ ausgefiltert werden, die zu einer Lösung mit unendlichen Kosten führen. Erfüllt P die Bedingung (3.5) nicht, wird diese Zeile mit $+\infty$ bewertet und ihre Berechnung beendet.

Angenommen, das Gegenteil wäre der Fall. Damit man $tab_Y(P)$ und $tab_X(S)$ verkettet, müssen $S \subseteq X$ und $P \subseteq Y$ dieselbe Knotenmenge Z aus dem Separator $X \cap Y$ haben:

$$S \cap (X \cap Y) = Z = P \cap (X \cap Y). \tag{3.9}$$

X kann aber mehr als eine Teilmenge haben, die der Gleichung (3.9) genügen. Man beachte, dass Z eine Teilmenge von Y ist, wobei Y zu dieser Zeit noch nicht bearbeitet ist. Man kann in diesem Fall Folgendes zeigen: Die Phase 1 konstruiert für jede $S \subseteq X$ den Teilgraphen $tab_X(S)$, so dass vor der Bearbeitung von Y keine Teilmenge Z von Y in $tab_X(S)$, also im Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = S$ zusammenhängt. (Diese Aussage, nämlich Lemma 3.6.3 werden wir allerdings nur für den Algorithmus im allgemeinen Fall zeigen.) Das heißt, egal für welche $S \subseteq X$ der Teilgraph $tab_X(S)$ gewählt und mit $tab_Y(P)$ verkettet wird, ist jeder Knoten $z \in Z(= S \cap (X \cap Y))$ in verschiedener Komponente des Teilgraphen $tab_X(S)$. Daher müssen die Knoten aus Z erst nach der Bearbeitung von Y zusammengehängt werden, und zwar in demselben Teilgraphen G_Y^X (Achtung: nicht G_X^Y). Dabei hat der Teilgraph $tab_X(S)$ für jede $S \subseteq X$ mit $S \cap (X \cap Y) = Z$ in G_Y^X dieselbe Knotenmenge Z und dieselbe leere Kantenmenge. Keine bietet einen Vorteil an. Daher liegt es nahe, den billigsten Teilgraphen $tab_X(S)$ auszuwählen. Anschließend wird $tab_Y(P)$ mit ausgewählter Zeile $tab_X(S)$ wie folgt verkettet:

- In die Zeigerliste der Zeile $tab_Y(P)$ wird ein Zeiger eingefügt, der auf $tab_X(S)$ zeigt;
- Die Kosten von $tab_Y(P)$ werden um die Kosten von $tab_X(S)$ erhöht.

Nachdem wir auf diese Weise jede tab_Y-Zeile mit einer tab_X-Zeile verkettet haben, beendet auch die Phase 2 von X. Somit ist die Bearbeitung des Baumknotens X abgeschlossen.

Es sei bemerkt, dass sowohl die Blätter als auch die innere Baumknoten, deren alle Kinder bearbeitet sind, auf gleiche Art und Weise bearbeitet werden. Der Grund dafür ist die oben erwähnte Eigenschaft des Algorithmus: Vor der Bearbeitung eines Baumknotens X gibt es keine Teillösung, in der eine Knotenteilmenge des Bags X zusammenhängt (Lemma 3.6.3). Das heißt, jeder Knoten $v \in X \setminus Y$ ist in verschiedener Komponente $H_X^Y(v)$ von H_X^Y, unabhängig davon ob X ein Blatt oder ein innerer Baumknoten ist. Daher ändert bei Phase 1 (die Konstruktion von H_X^Y) nichts: Die Komponente $H_X^Y(v)$ mit $v \in X \setminus Y$ muss solange wachsen, bis sie einen Knoten aus dem Separator $X \cap Y$ hat. Offensichtlich spielt auch für die Phase 2 keine Rolle, ob X ein Blatt oder ein innerer Baumknoten ist.

Der Algorithmus bearbeitet die Knoten von J eine nach dem anderen und erreicht irgendwann das Kind W der Wurzel R (Erinnerung: wegen der Idee 3.1.1 hat der Wurzel nur ein Kind). Wegen der Konstruktion besteht das Bag R aus nur einem Terminalknoten t und hat daher genau zwei Zeilen $tab_R(\{t\})$ und $tab_R(\{\})$, wobei die letztere die Bedingung (3.1) verletzt und daher unzulässig ist. Der Algorithmus bearbeitet W und fügt einen Zeiger in die Zeigerliste von $tab_R(\{t\})$, der auf den billigsten Teilgraphen H_W^R zeigt. Dieser Teilgraph ist nach Idee 3.1.1 ein (nicht unbedingt optimaler) Steinerbaum.

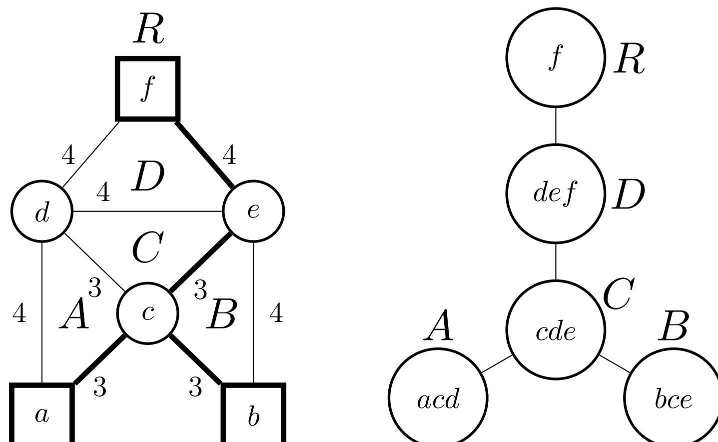


Abbildung 3.5: Eine Steinerbaum-Instanz und seine Baumzerlegung

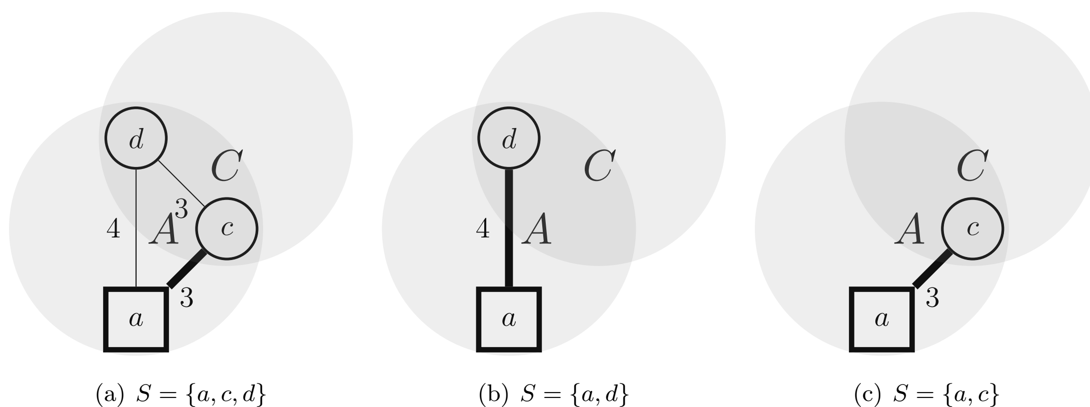
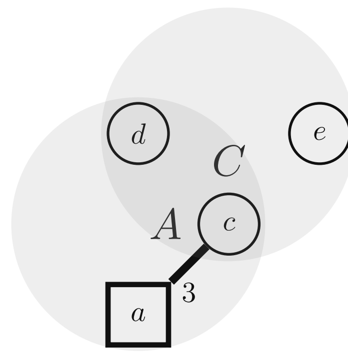


Abbildung 3.6: Phase 1 von A: Die Teilgraphen $G[S]$ und H_A^C (fette Kanten) mit $V(H_A^C) \cap A = S$

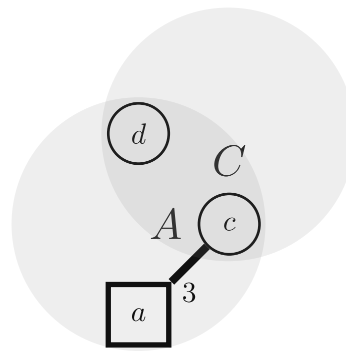
3.3.1 Ein Beispiel

Anhand der Steinerbaumproblem-Instanz aus der Abbildung 3.5 wollen wir das obige dynamische Programm vorstellen. Wir starten bei dem Blatt A.

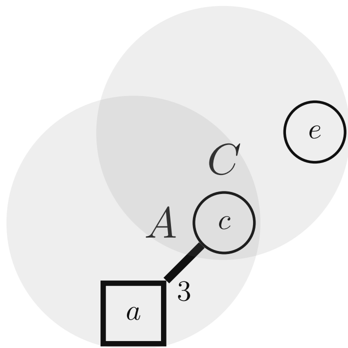
Phase 1 von A: Wir konstruieren H_A^C mit $V(H_A^C) \cap A = S$ für jede Teilmenge $S \subseteq A$, die die Bedingungen (3.1) und (3.4) erfüllt. Diese Bedingungen sind nur für drei Teilmengen von A erfüllbar: $S_1 := \{a, c, d\}$, $S_2 := \{a, d\}$, $S_3 := \{a, c\}$ (Abbildung 3.6). Für $S := S_1$ hat der Teilgraph $tab_A(S)$, also der Teilgraph H_A^C mit $V(H_A^C) \cap A = S$ initial drei Komponenten: $(\{a\}, \emptyset)$, $(\{c\}, \emptyset)$, $(\{d\}, \emptyset)$, wobei nur $K := (\{a\}, \emptyset)$ keinen Knoten aus dem Separator $A \cap C = \{c, d\}$ hat. Wir lassen die Komponente K soweit wachsen, bis er einen Knoten aus $S \cap (A \cap C) = \{c, d\}$ enthält. Nach der Erweiterung wird $K = (\{a, c\}, \{ac\})$ mit Kosten $c_{ac} = 3$ und $tab_A(S)$ erfüllt die Bedingung **H***. Der Teilgraph H_A^C besteht nun aus zwei



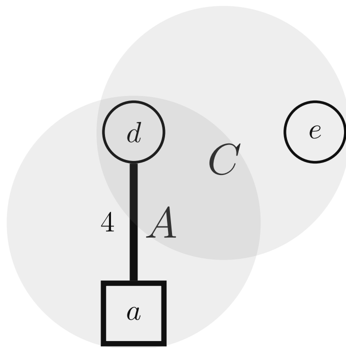
(a) $tab_C(P)$ für $P = \{c, d, e\}$ verkettet mit $tab_A(S)$ für $S = \{a, c, d\}$



(b) $tab_C(P)$ für $P = \{c, d\}$ verkettet mit $tab_A(S)$ für $S = \{a, c, d\}$



(c) $tab_C(P)$ für $P = \{c, e\}$ verkettet mit $tab_A(S)$ für $S = \{a, c\}$



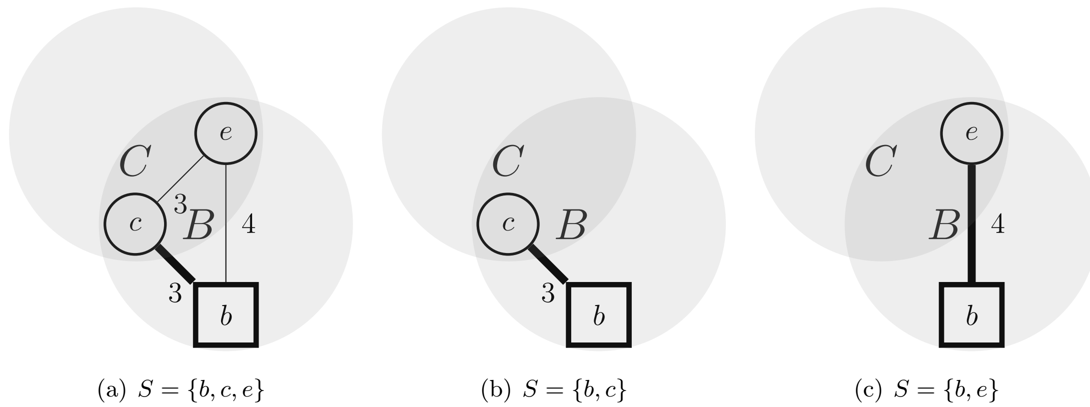
(d) $tab_C(P)$ für $P = \{d, e\}$ verkettet mit $tab_A(S)$ für $S = \{a, d\}$

Abbildung 3.7: Der Teillgraph $tab_C(P)$ nach der Phase 2 von A

Komponenten, nämlich $(\{d\}, \emptyset)$ und K (Abbildung 3.6(a)). Auf diese Weise berechnen wir $tab_A(S)$ auch für die restliche Teilmengen $S \in \{S_2, S_3\}$ (Abbildung 3.6).

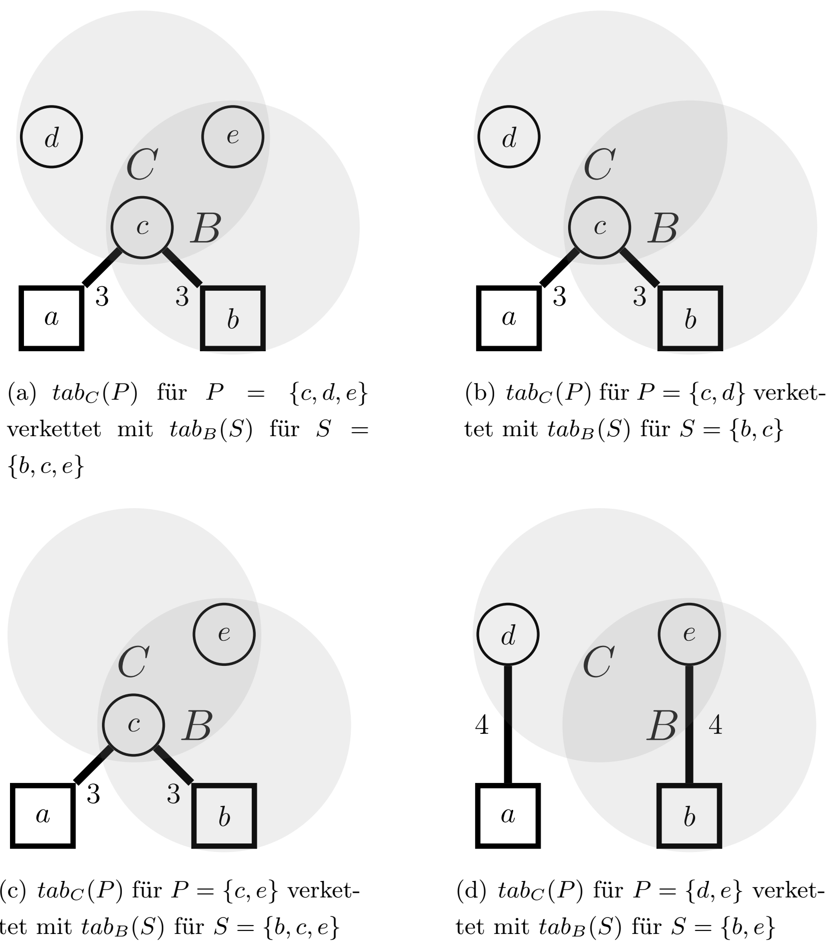
Phase 2 von A: Wir verketten die Teillösung $tab_C(P)$ für jede Teilmenge $P \subseteq C$ mit bester Teillösung $tab_A(S)$. Dabei betrachten wir nur die Teilmengen P, die die Bedingungen (3.1) und (3.5) erfüllen. Diese Bedingungen sind nur für die folgende Teilmengen erfüllbar: $P_1 := \{c, d, e\}$, $P_2 := \{c, d\}$, $P_3 := \{c, e\}$, $P_3 := \{d, e\}$. Betrachten wir beispielsweise $P_1 := \{c, d, e\}$. Falls $tab_C(P_1)$ mit einer Teilgraphen $tab_A(S)$ verkettet wird, müssen C, P_1, A und S die Bedingung (3.9) erfüllen. Es ist $Z := P_1 \cap (A \cap C) = \{c, d\}$. Für P_1 erfüllt nur $S_1 = \{a, c, d\}$ (siehe Phase 1 von A) diese Bedingung und ist daher der einzige Kandidat. Wir verketten den Teilgraphen $tab_C(P_1)$ daher mit $tab_A(S_1)$. Falls es mehrere Kandidaten gäbe, dann würden wir den billigsten wählen. Diese Situation ist in Abbildung 3.7(a) dargestellt. Auf diese Weise betrachten wir auch die restliche Teilmengen P_2, P_3 und P_4 (siehe Abbildung 3.7).

Bearbeitung von B: Die beiden Phasen laufen ähnlich zu A. Die Ergebnisse sind entsprechend in Abbildungen 3.8 und 3.9 zu sehen.



(a) $S = \{b, c, e\}$ (b) $S = \{b, c\}$ (c) $S = \{b, e\}$

Abbildung 3.8: Phase 1 von B: Die Teilgraphen $G[S]$ und H_B^C (fette Kanten) mit $V(H_B^C) \cap B = S$



(a) $tab_C(P)$ für $P = \{c, d, e\}$ verkettet mit $tab_B(S)$ für $S = \{b, c, e\}$

(b) $tab_C(P)$ für $P = \{c, d\}$ verkettet mit $tab_B(S)$ für $S = \{b, c\}$

(c) $tab_C(P)$ für $P = \{c, e\}$ verkettet mit $tab_B(S)$ für $S = \{b, c, e\}$

(d) $tab_C(P)$ für $P = \{d, e\}$ verkettet mit $tab_B(S)$ für $S = \{b, e\}$

Abbildung 3.9: Der Teillgraph $tab_C(P)$ nach der Phase 2 von B

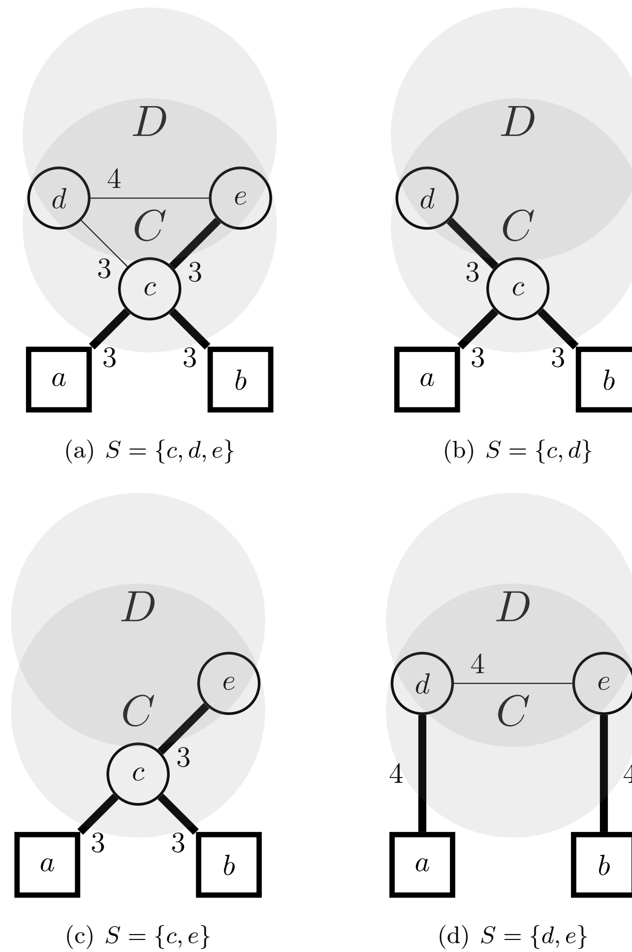
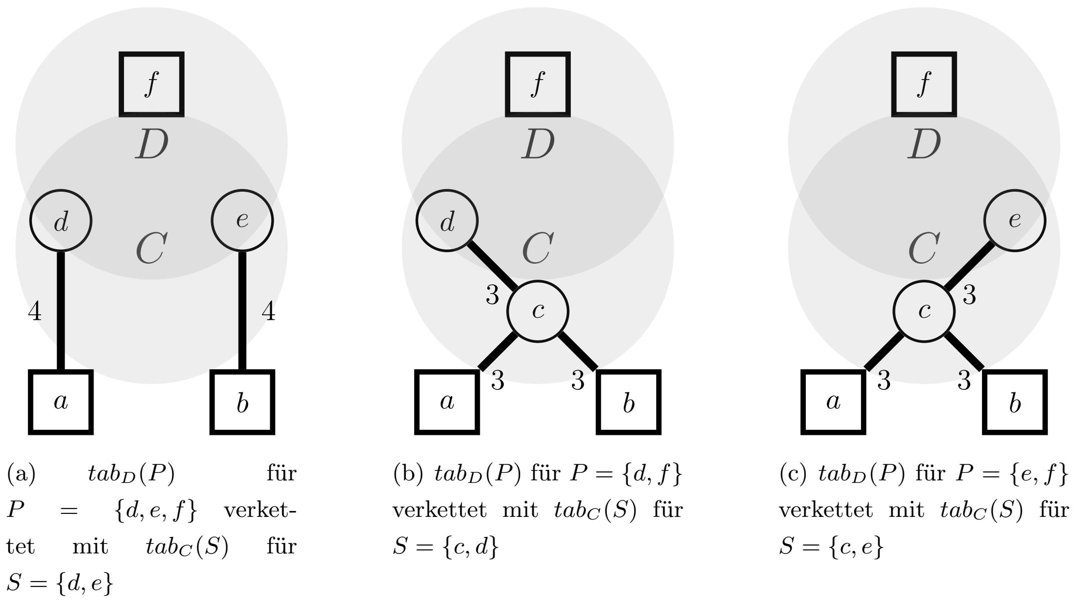


Abbildung 3.10: Phase 1 von C: Die Teilgraphen $G[S]$ und H_C^D (fette Kanten) mit $V(H_C^D) \cap C = S$

Phase 1 von C läuft bis auf die Teilmenge $S = \{d, e\}$ (Abbildung 3.10(d)) wie bisher. Für diese Teilmenge hat jede Komponente des Teilgraphen $tab_C(S)$ einen Knoten aus dem Separator $C \cap D$. Daher erfüllt dieser Teilgraph die Bedingung **H*** von alleine. Die Ergebnisse dieser Phase ist in Abbildung 3.10 zu sehen.

Phase 2 von C: Außerdem gibt es zwei Kandidaten die mit $tab_D(P)$ für $P := \{d, e, f\}$ verkettet werden könnten. Der Kandidat $tab_C(S)$ mit $S := \{c, d, e\}$ (Abbildung 3.10(a)) kostet 9, während $tab_C(S')$ mit $S' := \{d, e\}$ (Abbildung 3.10(d)) nur 8 kostet. Daher wird $tab_D(P)$ mit $tab_C(S')$ verkettet (Abbildung 3.11(a)). Die restliche Ergebnisse dieser Phase ist in Abbildung 3.11 zu sehen.

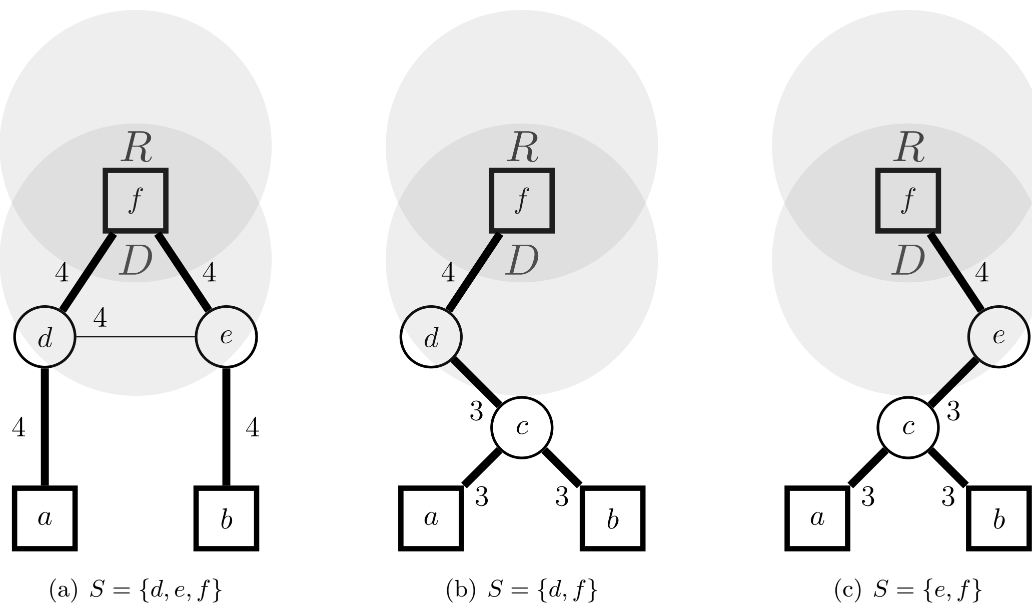


(a) $tab_D(P)$ für $P = \{d, e, f\}$ verkettet mit $tab_C(S)$ für $S = \{d, e\}$

(b) $tab_D(P)$ für $P = \{d, f\}$ verkettet mit $tab_C(S)$ für $S = \{c, d\}$

(c) $tab_D(P)$ für $P = \{e, f\}$ verkettet mit $tab_C(S)$ für $S = \{c, e\}$

Abbildung 3.11: Der Teillgraph $tab_D(P)$ nach der Phase 2 von C



(a) $S = \{d, e, f\}$

(b) $S = \{d, f\}$

(c) $S = \{e, f\}$

Abbildung 3.12: Phase 1 von D: Die Teilgraphen $G[S]$ und H_D^R (fette Kanten) mit $V(H_D^R) \cap D = S$

Bearbeitung von D: Phase 1 läuft wie bisher und konstruiert für jede Teilmenge $S \subseteq D$, die die Bedingungen (3.1) und (3.4) erfüllt, den Teilgraphen H_D^R mit $V(H_D^R) \cap D = S$ (Abbildung 3.12). Wegen der Konstruktion enthält der Separator $D \cap R$ nur einen Knoten f. Daher enthält jede Komponente von H_D^R den Knoten f. Somit ist H_D^R ein zusammenhängender Teilgraph von $G_D^R = G$. Außerdem enthält er jeden Terminalknoten von $G_D^R = G$. Daher ist jeder konstruierte H_D^R ein (nicht unbedingt optimaler) Steinerbaum. Wegen der Konstruktion hat die Tabelle tab_R nur eine Zeile $tab_R(P)$ mit $P := \{f\}$. Phase 2 speichert nun die billigste Teillösung H_D^R, beispielsweise H_D^R mit $V(H_D^R) \cap D = \{e, f\}$ (Abbildung 3.12(c)), in Zeile $tab_R(P)$. Der Teilgraph $tab_R(P)$ wird als ein Steinerbaum ausgegeben.

3.3.2 Schnelle Verkettung

Sei X ein Baumknoten mit Elter Y. Phase 1 von X hat eine Eigenschaft, die zu einer schnellen Verkettung führt, nämlich: *"Vor der Bearbeitung von Y gibt es keine Teillösung $tab_X(S)$ mit $S \subseteq X$, in dem eine Knotenteilmenge von Y zusammenhängt."* Diese Aussage (Lemma 3.6.3) werden wir nur für den Algorithmus im allgemeinen Fall beweisen.

Phase 2 von X sucht für jede Zeile $tab_Y(P)$ mit $P \subseteq Y$ eine Zeile $tab_X(S)$ mit $S \subseteq X$, die der Gleichung (3.9) genügt. Wegen der erwähnten Eigenschaft der Phase 1 muss diese Zeile möglichst billig sein. Naive Verkettung durchläuft die ganze Tabelle tab_X um die billigste Zeile $tab_X(S)$ zu finden. Da die beide Tabellen $O(2^{tw+1})$ Zeilen haben, braucht die naive Verkettung insgesamt $O\left((2^{tw+1})^2\right)$ Iterationen.

Die schnelle Verkettung verwaltet eine Variable *best*, wobei $best(Z)$ für $Z \subseteq X \cap Y$ eine Teilmenge $S \subseteq X$ angibt, so dass $tab_X(S)$ die billigste tab_X-Zeile mit $S \cap (X \cap Y) = Z$ ist.

Die Konstruktion dieser Variable findet in Phase 2 vor der Verkettung statt. Dazu wird die Variable *best* für jede $Z \subseteq X \cap Y$ mit $best(Z) := Z$ initialisiert. Diese Initialisierung braucht höchstens 2^{tw+1} Iterationen, da Z eine Teilmenge des Bags X ist, und ein Bag höchstens $tw + 1$ Knoten und somit höchstens 2^{tw+1} Teilmengen hat. Als Nächstes durchlaufen wir alle tab_X-Zeilen und aktualisieren dabei für jede Zeile $tab_X(S)$ mit $S \cap (X \cap Y) = Z$ die Variable $best(Z)$ wie folgt: Wir setzen $best(Z) := S$ falls $tab_X(S)$ billiger als $tab_X(best(Z))$ ist. Dieser Durchlauf hat höchstens 2^{tw+1} Iterationen. Die Berechnung der Variable $best(Z)$ für jede $Z \subseteq X \cap Y$ braucht somit höchstens $2^{tw+1} + 2^{tw+1} = O(2^{tw})$ Iterationen.

Die Variable $best(Z)$ ist korrekt berechnet. Der Grund dafür ist Folgendes: Setzt man $S := Z$ ein, so gilt $S \cap (X \cap Y) = Z$. Daher ist die Initialisierung $best(Z) := Z$ korrekt. Danach wird $best(Z)$ nur durch S mit $S \cap (X \cap Y) = Z$ aktualisiert. Da man alle tab_X-Zeilen $tab_X(S)$ betrachtet hat, bleibt keine Zeile vergessen. Daher hat $best(Z)$ die gewünschte Eigenschaft.

Als Nächstes wird die Verkettung gestartet und die Zeile $tab_Y(P)$ für jedes $P \subseteq Y$ mit $P \cap (X \cap Y) = Z$ mit der Zeile $tab_X(best(Z))$ verkettet. Für alle tab_Y-Zeilen werden höchstens 2^{tw+1} Iterationen benötigt.

Zusammen mit der Konstruktion der Variable *best* braucht die schnelle Verkettung insgesamt $O(2^{tw}) + 2^{tw+1} = O(2^{tw})$ Iterationen.

3.3.2 Lemma. *(Laufzeit) Der Algorithmus für den Spezialfall kann mit Hilfe der schnellen Verkettung in Zeit* $F + O\left(|V| \cdot 2^{tw} \cdot tw^2\right)$ *durchgeführt werden, wobei* F *eine Zufallsvariable mit Erwartungswert* $O\left(|V| \cdot 2^{tw} \cdot tw\right)$ *ist.*

Beweis. Nach Bemerkung 3.1.3 hat J höchstens $|V| + 1$ Baumknoten. Für jeden Baumknoten wird eine Tabelle mit höchstens 2^{tw+1} Zeilen erstellt.

Im Laufe des Algorithmus werden die Bedingungen (3.1), (3.4), (3.5) und (3.9) überprüft. Wir haben in Lemma 3.1.7 festgestellt, dass die Überprüfung der ersten drei Bedingungen einmalig in $O(|V| \cdot tw + tw^2)$, danach immer in $O(tw^2)$ Zeit durchgeführt werden kann. Man kann leicht sehen, dass die vierte Bedingung (3.9) analog zur Bedingung (3.4) auch in $O(tw^2)$ Zeit überprüft werden kann.

Die Bearbeitung eines Baumknotens X läuft in zwei Phasen. In Phase 1 wird für jede Zeile $tab_X(S)$ zuerst in $O(tw^2)$ Zeit (Lemma 3.1.7) überprüft, ob S die Bedingungen (3.1) und (3.4) erfüllt. Ist das der Fall, so wird im Teilgraphen $G[S]$, der höchstens $tw + 1$ Knoten hat, der Algorithmus von Prim durchgeführt. Das kostet $O(tw^2)$ Zeit. Eine tab_X-Zeile benötigt daher $O(tw^2 + tw^2) = O(tw^2)$ Zeit. Da tab_X höchstens 2^{tw+1} Zeilen hat, kann Phase 1 in $O\left(2^{tw} \cdot tw^2\right)$ Zeit durchgeführt werden.

In Phase 2 konstruiert die schnelle Verkettung zuerst die Variable $best(Z)$ für jede $Z \subseteq X \cap Y$. Diese Variable wird als *statisches perfektes Hashing* [7] realisiert, wobei jede Teilmenge $Z \subseteq X \cap Y$ ein Schlüssel der Hashtabelle *best* ist. Für s Schlüssel erstellt statisches perfektes Hashing eine Tabelle in erwarteter Zeit $O(s)$, so dass anschließend jeder Schlüssel in konstanter Zeit gefunden wird. Wir dürfen statisches perfektes Hashing benutzen, da alle Schlüssel von vornerein bekannt sind und jeder Schlüssel nur einmal am Anfang eingefügt und dann nie gelöscht wird. Zuerst wird $best(Z)$ für jede $Z \subseteq X \cap Y$ mit $best(Z) := Z$ initialisiert, das heißt, es wird die Hashtabelle *best* mit $O(2^{tw})$ Elementen in erwarteter Zeit $O(2^{tw} \cdot tw)$ erstellt. Der Faktor $O(tw)$ wird dabei für die Berechnung des Hashwertes benötigt, da Z nicht ein Knoten sondern eine Menge mit $O(tw)$ Knoten ist. Wir werden zur Erstellung der Hashtabellen benötigte Zeit getrennt zählen. Da wir $O(|V|)$ Baumknoten bearbeiten, werden wir insgesamt $O(|V|)$ Hashtabellen in erwarteter Zeit $O(|V| \cdot 2^{tw} \cdot tw)$ erstellen.

Als Nächstes werden alle $O(2^{tw})$ Zeilen der Tabelle tab_X durchlaufen und eventuell die Aktualisierung $best(Z) := S$ für ein $S \subseteq X$ mit $S \cap (X \cap Y) = Z$ durchgeführt. Da die Berechnung von $S \cap (X \cap Y)$ in $O(tw^2)$ (Lemma 3.1.7) Zeit durchgeführt und $best(Z)$ in

$O(tw)$ (wegen der Berechnung des Hashwertes) Zeit zugegriffen werden kann, benötigt die Aktualisierung der Hashtabelle *best* insgesamt $O\left(2^{tw} \cdot (tw^2 + tw)\right) = O(2^{tw} \cdot tw^2)$ Zeit.

Als Nächstes wird für jede tab_Y-Zeile $tab_Y(P)$ in $O(tw^2)$ Zeit (Lemma 3.1.7) überprüft, ob P die Bedingung (3.5) erfüllt. Ist das der Fall, so wird $Z := P \cap (X \cap Y)$ in $O(tw^2)$ Zeit (siehe oben) berechnet und $tab_Y(P)$ in $O(tw)$ (Berechnung des Hashwertes) Zeit mit der Zeile $tab_X(best(Z))$ verkettet. Die Verkettung einer Zeile $tab_Y(P)$ kostet somit $O(tw^2 + tw^2 + tw) = O(tw^2)$ Zeit. Für die Verkettung aller tab_Y-Zeilen werden somit insgesamt $O\left(2^{tw} \cdot tw^2\right)$ Zeit benötigt. Mit Hilfe der schnellen Verkettung kann somit auch die Phase 2 in $O\left(2^{tw} \cdot tw^2\right)$ Zeit durchgeführt werden.

Wir fassen zusammen: Mit Hilfe der schnellen Verkettung können alle $O(|V|)$ Baumknoten von J in Zeit $O\left(|V| \cdot 2^{tw} \cdot tw^2\right) + F + O(|V| \cdot tw + tw^2)$ bearbeitet werden. Dabei ist F eine Zufallsvariable mit Erwartungswert $O(|V| \cdot 2^{tw} \cdot tw)$, die zur Erstellung aller Hashtabellen *best* benötigten Zeit angibt. Der Term $O(|V| \cdot tw + tw^2)$ steht für die einmalige Kosten zur Berechnung der Bedingungen (3.1), (3.4), (3.5). Daraus folgt die Behauptung.□

3.4 Relevante Pfade

3.4.1 Trick mit künstlicher Kante

Der Algorithmus für den Spezialfall funktioniert leider nicht bei allgemeinen Graphen. Das Problem liegt an Phase 1. Hier versucht man die Komponenten einer Teillösung in einem Teilgraphen von G wachsen zu lassen. Aber ein Teilgraph ist nicht unbedingt zusammenhängend, wenn auch G selbst zusammenhängt. Daher hatten wir die obige Voraussetzung.

Diese Situation ist in Abbildung 3.13 dargestellt. Hier ist ein Graph und seine Baumzerlegung zu sehen. Die gestrichelte Kanten gehören nicht zum Graphen, sondern zeigen quasi die Grenzen der Bags. Wir betrachten die Konstruktion des Teilgraphen H_B^R, der alle Knoten aus B enthalten soll. Phase 1 versucht in $G[B]$ den Knoten c mit dem Separator Knoten $d \in B \cap R$ zu verbinden. Das geht aber nicht, da die beide Knoten in $G[B]$ nicht zusammenhängen.

Obiges Problem kann leicht beseitigt werden. Wir können ähnlich zur Distanznetzwerkheuristik [12, §6] vorgehen. Da der Graph G zusammenhängend ist, gibt es in G einen einfachen (c, d)-Pfad. Wir können künstlich eine Kante cd mit Kosten c_{cd} in $G[B]$ einfügen, wobei c_{cd} gleich zu der Länge des kürzesten (c, d)-Pfads in G ist. Man kann mit diesem Trick jeden Teilgraphen von G zusammenhängend machen und der obige Algorithmus wird wieder funktionieren. Anschließend müssen natürlich die künstlichen Kanten durch die original Pfade ersetzt werden. In unserem Beispiel ist $P := c, e, d$ der kürzeste (c, d)-Pfad mit der Länge $4(= 2 + 2)$. Wir erzeugen eine künstliche Kante cd mit Kosten 4 und verbinden die beide Knoten(Abbildung 3.14(a)). Am Ende wird diese Kante durch den Pfad

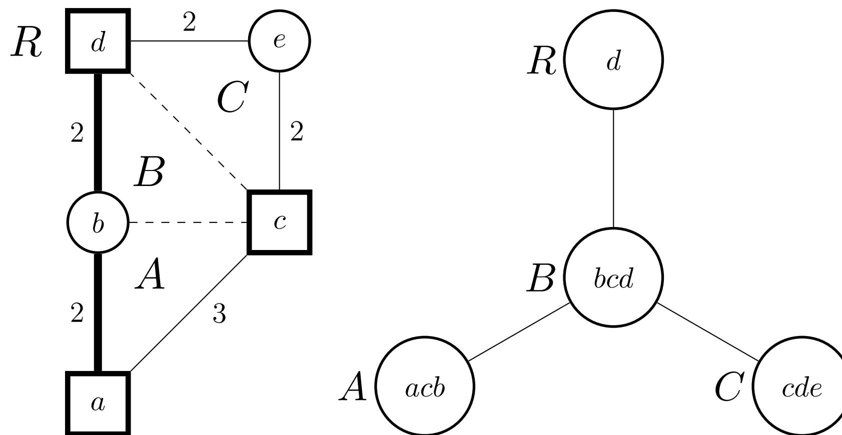
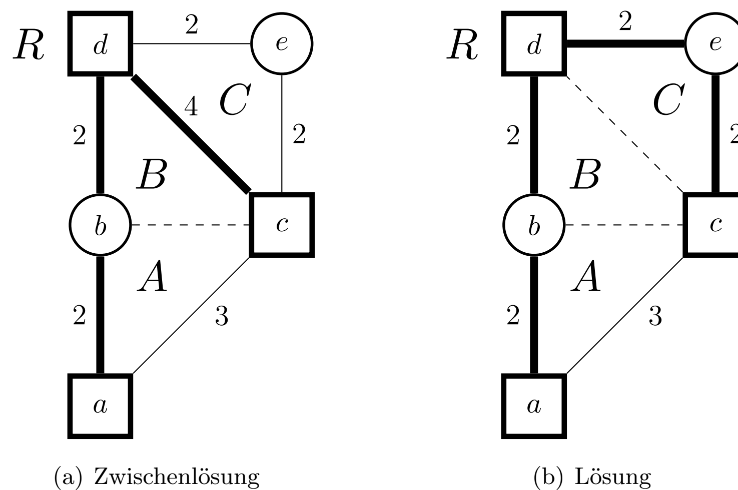


Abbildung 3.13: Graph und seine Baumzerlegung



(a) Zwischenlösung (b) Lösung

Abbildung 3.14: Trick mit künstlicher Kante *cd*

P ersetzt und wir bekommen einen (*nicht unbedingt optimalen*) Steinerbaum mit Kosten $8(= 2 + 2 + 2 + 2)$ (Abbildung 3.14(b)).

Wir betrachten die obige Situation aus einem anderen Blickwinkel. Bevor wir eine neue Kante cd künstlich in G zugefügt hatten, gab es in G keine Kante zwischen den Knoten c und d. Stellen wir uns vor, dass G schon eine echte Kante cd mit Kosten $c_{cd} = +\infty$ hatte und diese echte Kante durch eine künstliche Kante cd mit Kosten $c'_{cd} < c_{cd}$ ersetzt wäre. Könnte es ebenfalls sinnvoll sein eine echte Kante cd mit Kosten $c_{cd} < +\infty$ durch eine künstliche Kante cd mit Kosten $c'_{cd} < c_{cd}$ zu ersetzen. Diese Überlegung führt zur folgenden Definition.

3.4.1 Definition. (Künstliche Kanten) Eine Kante uv mit Kosten c ($c = +\infty$ auch möglich) heißt künstlich, wenn G keine Kante uv mit Kosten $c_{uv} = c$ hat.

3.4.2 Bemerkung. (Kanten mit unendlichen Kosten) Natürlich sind wir an einem Steinerbaum mit endlichen Kosten interessiert. Ein Steinerbaum mit endlichen Kosten enthält keine Kante mit unendlichen Kosten. Daher dürfen wir jede Kante uv mit unendlichen Kosten aus G entfernen. Wir dürfen auch in G eine neue Kante uv mit unendlichen Kosten einfügen, falls in G keine Kante zwischen den Knoten u und v verläuft.

3.4.2 Relevante Pfade

Da alle Kantenkosten positiv sind, können alle in G kürzeste Pfade beispielsweise mit Dijkstra oder Floyd-Warshal Algorithmus in Zeit $O(|V|^3)$ berechnet werden. Aus zwei Gründen werden wir anders vorgehen. Erstens ist der obige Ansatz nicht linear in $|V|$. Zweitens, um die schnelle Verkettung aus Abschnitt 3.3.2 benutzen zu können, interessieren uns die kürzesten Pfade mit bestimmter Eigenschaft. Sei $(J, \mathcal{X})$ eine Baumzerlegung mit Baumweite tw des Graphen G, wobei J gemäß der Idee 3.1.1 mit R gewurzelt ist. Sei X ein Baumknoten mit Elter Y. Der Teilgraph H_X^Y, der aus dem Bag X nur die Knotenteilmenge S enthält (also $X \setminus S$ nicht enthält), liegt im Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$. Ein in G kürzester Pfad liegt nicht unbedingt in diesem Teilgraphen. Für die Konstruktion von H_X^Y sind also nur die Pfade relevant, die völlig in diesem Teilgraphen liegen. Insbesondere interessiert uns die (u, v)-Pfade mit $u, v \in S$. Man kann diese Beschränkungen effizient benutzen und alle benötigte relevante Pfade in erwarteter Zeit $O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ berechnen.

Wir werden den Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$ und seine Pfade als *relevant* für H_X^Y mit $V(H_X^Y) \cap X = S$ bezeichnen.

Algorithmus für relevante Pfade. Wegen der Bemerkung 3.4.2 dürfen wir annehmen, dass G keine Kante mit unendlichen Kosten enthält (gegebenenfalls entfernen wir sie).

Sei $(J, \mathcal{X})$ eine Baumzerlegung mit Baumweite tw des Graphen G, wobei J gemäß der Idee 3.1.1 mit R gewurzelt ist. In Initialisierungsphase wird für jedes Bag $X \neq R$ mit Elter Y eine Tabelle $path_X$ erzeugt, die für jede Teilmenge $S \subseteq X$ eine Zeile $path_X(S)$ hat. In dieser Zeile werden die relevante (u, v)-Pfade P mit $u, v \in S$ verwaltet, die im Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$ am kürzesten sind. Da das Bag R wegen der Idee 3.1.1 genau einen Knoten enthält, haben wir R ignoriert.

Die zentrale Beobachtung des Algorithmus ist Folgendes: Jeder einfache (u, v)-Pfad in $G[V_X^Y \setminus (X \setminus S)]$ mit $u, v \in S$ kann in eine Sequenz der Teilpfade $P_1, \cdots, P_l$ so zerlegt werden, dass der Teilpfad P_i für jeden $i \in \{1, \cdots, l\}$, entweder völlig in $G[S]$ ist, oder nur seine Endknoten (keinen weiteren Knoten und keine Kante) in $G[S]$ hat. Das führt zur folgenden Idee. Konstruiere einen vollständigen Hilfsgraphen $G'[S]$, so dass jeder Pfad, der in $G[S]$ liegt, auch in $G'[S]$ liegt und für jeden kürzesten Pfad P in $G[V_X^Y \setminus (X \setminus S)]$, dessen nur Endknoten u, v in $G[S]$ sind, hat $G'[S]$ eine *künstliche Kante uv* mit Kosten $c'_{uv} = c(P)$. Man kann dann zeigen, dass $G[V_X^Y \setminus (X \setminus S)]$ für jedes Knotenpaar $u, v \in S$ genau dann einen (u, v)-Pfad P mit Länge d hat, wenn $G'[S]$ einen (u, v)-Pfad P' mit

Länge d hat. Dann können wir d beispielsweise mit Dijkstra oder Floyd-Warshal in $G'[S]$ anstatt in $G[V_X^Y \setminus (X \setminus S)]$ berechnen. Das ist viel effizienter. Denn $G'[S]$ hat $|S| = O(tw)$ Knoten, $G[V_X^Y \setminus (X \setminus S)]$ dagegen $O(|V|)$ Knoten.

P' kann also höchstens tw Kanten haben, P dagegen $O(|V|)$ Kanten. P' wird dann wie folgt aussehen: Ist ein Teilpfad P_i von P in $G[S]$, dann enthält P' den Teilpfad P_i auch. Sind nur die Endknoten w, z von P_i in $G[S]$, so wird P' anstelle des Teilpfads P_i eine künstliche Kante wz mit Kosten $c'_{wz} = c(P_i)$ enthalten. Wir definieren dementsprechend einen Zeiger $p(wz) := P_i$. Aus Effizienzgründen werden wir nicht P sondern P' speichern. Da ein kürzester relevante Pfad von dem dazugehörenden relevanten Teilgraphen abhängig ist, bezeichnen wir den (u, v)-Pfad, der in Zeile $path_X(S)$ gespeichert wird (also nicht P sondern P'), mit $uvXS$ und seine Länge mit $d(uvXS)$.

Der Zeiger $p(uv)$ zeigt auf einen (u, v)-Pfad P, der in $G[V_X^Y \setminus (X \setminus S)]$ liegt und nur seinen Endknoten (keinen weiteren Knoten und keine Kante) in $G[S]$ hat. Solch einen Pfad kann es nur dann geben, wenn $V_X^Y \setminus (X \setminus S) \neq S$ ist, dass heißt, wenn X kein Blatt ist. Denn wenn X ein Blatt ist, dann gilt $V_X^Y = X$. Daher gilt $p(uv) :=$Null für jedes $u, v \in S$ in einer Zeile $path_X(S)$, falls X kein Kind hat.

Das dynamische Programm bearbeitet J Bottom-Up startend bei den Blätter hoch zur Wurzel. Genauer gesagt, dauert die Bearbeitung bis zum Kind W der Wurzel R. (Erinnerung: wegen der Idee 3.1.1 hat der Wurzel nur ein Kind.)

Sei X ein Blatt. Für jede Teilmenge $S \subseteq X$ konstruieren wir einen vollständigen Hilfsgraphen $G'[S]$. Die Kantenkosten c' in $G'[S]$ sind wie folgt definiert: $c'_{uv} := c_{uv}$ falls $G[S]$ die Kante uv hat, sonst $+\infty$. Anschließend wird der Pfad $uvXS$ und seine Länge für jedes Knotenpaar $u, v \in S$ in $G'[S]$ berechnet und in $path_X(S)$ gespeichert. Außerdem speichern wir $p(uv) :=$Null für jedes Knotenpaar $u, v \in S$.

Sei X ein innerer Baumknoten, dessen Kinder bereits bearbeitet wurden. Für jede Teilmenge $S \subseteq X$ konstruieren wir einen vollständigen Hilfsgraphen $G'[S]$. Die Kantenkosten c' in $G'[S]$ werden wie folgt initialisiert: $c'_{uv} := c_{uv}$ falls $G[S]$ die Kante uv hat, sonst $+\infty$. Außerdem setzen wir $p(uv) :=$Null für jedes Knotenpaar $u, v \in S$. Als Nächstes aktualisieren wir c'_{uv} für jedes Knotenpaar $u, v \in S \cap (W \cap X)$ für jedes Kind W von X wie folgt:

$$c'_{uv} := \min\big(\, c'_{uv},\ d(uvWQ) \text{ mit } Q := W \setminus (X \setminus S) \,\big) \tag{3.10}$$

Diese Aktualisierung kann eventuell c'_{uv} ändern. In diesem Fall wissen wir, dass die $G'[S]$ Kante uv eine künstliche Kante ist, die den kürzesten relevanten Pfad $uvWQ$ vertritt. Wir aktualisieren $p(uv) := uvWQ$. Nachdem $G'[S]$ konstruiert wurde, wird der Pfad $uvXS$ und seine Länge für jedes Knotenpaar $u, v \in S$ in $G'[S]$ berechnet und in $path_X(S)$ gespeichert. Außerdem speichern wir $p(uv)$ für jedes Knotenpaar $u, v \in S$.

3.4.3 Lemma. *(Korrektheit) Gegeben seien ein Baumknoten X mit Elter Y, ein $S \subseteq X$ und ein Knotenpaar $u, v \in S$. Obiger Algorithmus berechnet einen Pfad $uvXS$ mit Länge*

$d < +\infty$, genau dann, wenn es einen in $G[V_X^Y \setminus (X \setminus S)]$ kürzesten (u, v)-Pfad P mit Kosten d gibt, der jede echte Kante von $uvXS$ enthält und anstelle jeder künstlichen Kante wz mit Kosten c'_{wz} von $uvXS$ einen (w, z)-Pfad mit Kosten c'_{wz} enthält.

Beweis. Der Algorithmus berechnet den Pfad $uvXS$ im Hilfsgraphen $G'[S]$ anstelle in $G[V_X^Y \setminus (X \setminus S)]$, und zwar mit Dijkstra oder Floyd-Warshal. Wir zeigen Folgendes:

i) Der Hilfsgraph $G'[S]$ hat einen (u, v)-Pfad mit Kosten $d < +\infty$ genau dann, wenn $G[V_X^Y \setminus (X \setminus S)]$ einen (u, v)-Pfad mit Kosten d enthält.

ii) Die Kanten in $G'[S]$ haben positive Kosten.

Der Grund für diese Aussagen ist Folgendes: Aus der Aussage *i)* folgt, dass, wenn der kürzeste (u, v)-Pfad P mit Kosten d in $G[V_X^Y \setminus (X \setminus S)]$ existiert, dann hat auch $G'[S]$ einen Pfad $uvXS$ mit Kosten d. Aus der Aussage *ii)* folgt, dass $uvXS$ beispielsweise mit Dijkstra berechnet werden kann. Außerdem folgt aus *i)*, dass, wenn, P nicht existiert, dann kann es auch in $G'[S]$ keinen Pfad $uvXS$ mit Kosten $d < +\infty$ geben. Darüber hinaus, folgt aus *i)*, dass für jede künstliche Kante wz mit Kosten c'_{wz} von $G'[S]$, somit auch von $uvXS$ (da $uvXS$ ein Teilgraph von $G'[S]$ ist), einen (w, z)-Pfad mit Kosten c'_{wz} in $G[V_X^Y \setminus (X \setminus S)]$ gibt. Daraus folgt die Behauptung des Lemmas.

i) Das zeigen wir mit vollständiger Induktion über die Baumknoten:

IA: Sei X ein Blatt. In diesem Fall ist $V_X^Y = X$ und daher gilt:

$$G[V_X^Y \setminus (X \setminus S)] = G[X \setminus (X \setminus S)] = G[S].$$

"⇒": Zuerst zeigen wir, dass es für jeden (u, v)-Pfad P' mit $c'(P') < +\infty$ in $G'[S]$ einen (u, v)-Pfad P mit $c(P) \leq c'(P')$ in $G[S]$ gibt. Da $c'(P') < +\infty$ ist, hat P' keine Kante mit unendlichen Kosten. Wegen der Konstruktion ist jede Kante von $G'[S]$, die endliche Kosten hat, auch in $G[S]$ enthalten. Daher ist jede Kante des Pfads P' in $G[S]$. Dann ist $P := P'$ ein Pfad mit $c(P) \leq c'(P')$ in $G[S]$.

"⇐": Als Nächstes zeigen wir, dass es für jeden (u, v)-Pfad P mit $c(P) < +\infty$ in $G[S]$ einen (u, v)-Pfad P' mit $c'(P') \leq c(P)$ in $G'[S]$ gibt. Da $G[S]$ ein Teilgraph von $G'[S]$ ist, liegt P auch in $G'[S]$. Dann ist $P' := P$ ein Pfad mit $c'(P') \leq c(P)$ in $G'[S]$.

IS: Falls X ein Blatt ist, kann die Korrektheit analog zu **IA** gezeigt werden. Angenommen, das Gegenteil wäre der Fall. Seien $W_1, \cdots, W_l$ die Kinder von X. Sei X ein innerer Baumknoten mit Kindern $W_1, \cdots, W_l$. $G[V_X^Y \setminus (X \setminus S)]$ besteht aus $G[X \setminus (X \setminus S)] = G[S]$ und $G[V_W^X \setminus (X \setminus S)]$ für jede $W \in W_1, \cdots, W_l$.

"$\Rightarrow$": Zuerst zeigen wir, dass es für jeden (u,v)-Pfad P' mit $c'(P') < +\infty$ in $G'[S]$ einen (u,v)-Pfad P mit $c(P) \leq c'(P')$ in $G[V_X^Y \setminus (X \setminus S)]$ gibt. Da $c'(P') < +\infty$ ist, hat P' keine Kante mit unendlichen Kosten.

Es gibt für jede Kante wz mit $c'_{wz} < +\infty$ des Hilfsgraphen $G'[S]$ ein Pfad P mit $c(P) \leq c'_{wz}$ in $G[V_X^Y \setminus (X \setminus S)]$. Der Grund dafür ist Folgendes. Die Kante wz ist entweder eine echte Kante in $G[S]$ (wegen der Initialisierung von $G'[S]$) oder ist wegen der Gleichung (3.10) eine künstliche Kante mit Kosten $c'_{wz} = d(wzWQ)$ für ein Kind W von X und $Q = W \setminus (X \setminus S)$. Da W ein Kind von X ist, ist $path_W$, somit auch $wzWQ$ vor $path_X$ berechnet. Nach Induktionsvoraussetzung gibt es dann ein (w,z)-Pfad in $G[V_W^X \setminus (X \setminus S)]$ mit Kosten $d(wzWQ) = c'_{wz}$. Da $G[S]$ und $G[V_W^X \setminus (X \setminus S)]$ die Teilgraphen von $G[V_X^Y \setminus (X \setminus S)]$ sind, folgt die Behauptung. Das gilt für jede Kante des Pfads P', da P' nur $G'[S]$ Kanten hat.

Sei $P' = e_1, \cdots, e_l$ und P_i für $i \in \{1, \cdots, l\}$ zu e_i entsprechender Pfad in $G[V_X^Y \setminus (X \setminus S)]$. Dann ist $P := P_1, \cdots, P_l$ ein $G[V_X^Y \setminus (X \setminus S)]$ Pfad, da die Teilpfaden $P_1, \cdots, P_l$ in $G[V_X^Y \setminus (X \setminus S)]$ liegen. Außerdem gilt:

$$c(P) = c(P_1) + \cdots + c(P_l) \leq c'_{e_1} + \cdots + c'_{e_l} = c'(P').$$

"$\Leftarrow$": Als Nächstes zeigen wir, dass es für jeden (u,v)-Pfad P mit $c(P) < +\infty$ in $G[V_X^Y \setminus (X \setminus S)]$ einen (u,v)-Pfad P' mit $c'(P') \leq c(P)$ in $G'[S]$ gibt.

Wir erinnern uns an die zentrale Beobachtung des Algorithmus. Jeder einfache (u,v)-Pfad in $G[V_X^Y \setminus (X \setminus S)]$ mit $u, v \in S$ kann in eine Sequenz der Teilpfade $P_1, \cdots, P_l$ so zerlegt werden, dass der Teilpfad P_i für jeden $i \in \{1, \cdots, l\}$, entweder völlig in $G[S]$ ist, oder nur seine Endknoten in $G[S]$ hat. Jeder Teilpfad P_i, dessen Kanten alle in $G[S]$ sind, liegt auch in $G'[S]$. Es reicht zu zeigen, dass für jeden Teilpfad $P_i := w_1, \cdots, w_j$, für den nur die Endknoten w_1, w_j in $G[S]$ sind, der Hilfsgraph $G'[S]$ eine Kante w_1w_j mit Kosten $c'_{w_1w_j} \leq c(P_i)$ enthält. Denn daraus folgt, dass es in $G'[S]$ einen (u,v)-Pfad P' gibt, der jeden in $G[S]$ liegenden Teilpfad von P enthält und anstelle jedes nicht in $G[S]$ liegenden Teilpfads P_i, dessen nur die Endknoten w_1, w_j in $G[S]$ sind, eine Kante w_1w_j mit $c'_{w_1w_j} \leq c(P_i)$ enthält. Und daraus folgt: $c'(P') \leq c(P)$.

Sei $P_i := w_1, \cdots, w_j$ für ein $1 \leq i \leq l$ ein Teilpfad von P, der nur seine Endknoten in $G[S]$ hat. Der Pfad P_i hat mindestens drei Knoten, sonst würde P_i nur aus zwei Knoten w_1 und w_j und einer Kante w_1w_j bestehen. Dann wäre aber außer seinen Endknoten w_1, w_j auch die Kante w_1w_j in $G[S]$. Daher hat P die Form $P := w_1, w_2, \cdots, w_j$ mit $j \geq 3$. Dann ist w_2 nicht in $G[S]$.

Sei $w_2 \in V_W^X$ für ein Kind W von X. Dann liegt P_i völlig in $G[V_W^X \setminus (X \setminus S)]$. Der Grund dafür ist Folgendes. Weil $w_2, \cdots, w_{j-1}$ ein Teilpfad von P ist und P keinen Knoten

aus $X \setminus S$ hat, hat $w_2, \cdots, w_{j-1}$ keinen $X \setminus S$ Knoten. Da der Teilpfad $w_2, \cdots, w_{j-1}$ aber auch keinen S Knoten hat, hat er überhaupt keinen Knoten in X. Insbesondere gilt $w_2 \in V_W^X \setminus X$. Da $X \neq R$ und $W \neq R$ ist, ist $W \cap X$ wegen der Bemerkung 3.1.2 ein Separator. Daher enthält jeder Pfad $w_2, \cdots, w'$ mit $w' \in V_X^W$ (Achtung: nicht V_W^X) mindestens einen Knoten aus $W \cap X$ somit auch aus X. Umgekehrt heißt das, wenn der Pfad $w_2, \cdots, w'$ keinen Knoten aus X hat, liegt w' nicht in V_X^W, sondern in $V_W^X \setminus X$. Da der Pfad $w_2, \cdots, w'$ für keinen Knoten $w' \in \{w_2, \cdots, w_{j-1}\}$ einen Knoten aus X hat, liegt jeder $w' \in \{w_2, \cdots, w_{j-1}\}$ in $V_W^X \setminus X$. Wegen $V_W^X \setminus X \subseteq V_W^X \setminus (X \setminus S)$ liegen die Knoten $w_2, \cdots, w_{j-1}$ in $G[V_W^X \setminus (X \setminus S)]$. Wir zeigen, dass die Endknoten w_1 und w_j des Pfads P_i auch in diesem Teilgraphen sind. Wegen der Bedingung **tw2** muss die Kante $w_1 w_2$ in irgendeinem Bag enthalten sein. Da der Knoten w_2 in $V_W^X \setminus X$ ist, ist er nicht im Separator $W \cap X$, somit auch nicht in V_X^W (Achtung: nicht V_W^X). Es bleibt nur ein Ort für die Kante $w_1 w_2$ übrig: sie muss in $G[V_W^X]$ sein. Da $w_1 w_2$ eine Kante des Pfads P ist und P keinen Knoten aus $X \setminus S$ hat, ist diese Kante in $G[V_W^X \setminus (X \setminus S)]$. Daher ist w_1 auch in $G[V_W^X \setminus (X \setminus S)]$. Analoges kann man auch für die letzte Kante $w_{j-1} w_j$, beziehungsweise für den Knoten w_j zeigen. Somit liegt (w_1, w_j)-Pfad P_i wie behauptetet völlig im Teilgraphen $G[V_W^X \setminus (X \setminus S)]$.

Wegen der Gleichung (3.10) enthält der Hilfsgraph $G'[S]$ eine Kante $w_1 w_j$ mit Kosten $c'_{w_1 w_j} \leq d(w_1 w_j W Q)$ mit $Q := W \setminus (X \setminus S)$. Da W ein Kind von X ist, ist $path_W$, somit auch $wzWQ$ vor $path_X$ berechnet. Nach Induktionsvoraussetzung ist dann $d(w_1 w_j W Q)$ die Länge des in $G[V_W^X \setminus (X \setminus S)]$ kürzesten Pfads, daher ist $d(w_1 w_j W Q) \leq c(P_i)$. Daraus folgt: $c'_{w_1 w_j} \leq c(P_i)$.

ii) Wegen der Konstruktion hat $G'[S]$ keine Kante e mit Kosten $c'_e = -\infty$. Jede $G'[S]$ Kante e mit Kosten $c'_e = +\infty$ ist positiv. Es bleibt nur noch zu zeigen, dass jede $G'[S]$ Kante e mit $c'_e < +\infty$ auch positiv ist. Zum Widerspruchsbeweis nehmen wir an, $G'[S]$ eine Kante uv mit $c'_{uv} \leq 0$ hat. Wir haben schon gezeigt, dass dann im Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$ einen (u, v)-Pfad P mit Kosten $c(P) \leq c'_{uv} \leq 0$ gibt. Ein (u, v)-Pfad P hat mindestens eine Kante. Da G, somit auch sein Teilgraph $G[V_X^Y \setminus (X \setminus S)]$ nur Kanten mit positiven Kosten hat und P mindestens eine Kante enthält, ist $c(P) > 0$. Widerspruch! □

3.4.4 Lemma. *(Laufzeit) Die Laufzeit des Algorithmus für relevante Pfade beträgt* $F + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$, *wobei* F *eine Zufallsvariable mit Erwartungswert* $O\left(|V| \cdot 2^{tw} \cdot tw\right)$ *ist.*

Beweis. Nach Bemerkung 3.1.3 hat J höchstens $|V| + 1$ Baumknoten. Für jeden Baumknoten $X \neq R$ wird eine Tabelle $path_X$ mit höchstens 2^{tw+1} Zeilen erstellt. Die Tabelle $path_X$ wird als *statisches perfektes Hashing* [7] realisiert, wobei die Teilmenge $S \subseteq X$ der Schlüssel der Zeile $path_X(S)$ ist. Für s Schlüssel erstellt statisches perfektes Hashing eine Tabelle in erwarteter Zeit $O(s)$, so dass anschließend jeder Schlüssel in konstanter Zeit gefunden wird. Wir dürfen statisches perfektes Hashing benutzen, da alle Schlüssel von

vornerein bekannt sind und jeder Schlüssel nur einmal am Anfang eingefügt und dann nie gelöscht wird. Die Tabelle $path_X$ wird in erwarteter Zeit $O(2^{tw} \cdot tw)$ erstellt. Der Faktor $O(tw)$ wird dabei für die Berechnung des Hashwertes eines Schlüssels $S \subseteq X$ benötigt, da S nicht ein Knoten sondern eine Menge mit $O(tw)$ Knoten ist. Alle $O(|V|)$ Hashtabellen können dann in erwarteter Zeit $O(|V| \cdot 2^{tw} \cdot tw)$ erstellt werden.

In jeder Zeile $path_X(S)$ einer Tabelle $path_X$ wird ein Hilfsgraph $G'[S]$ konstruiert. Im schlimmsten Fall ist X ein innerer Baumknoten mit Kindern $W_1, \cdots, W_l$. In diesem Fall muss die Adjazenzliste $W_1, \cdots, W_l$ von X durchlaufen werden, um die Kantenkosten von $G'[S]$ zu berechnen. Wir zeigen, dass für die Konstruktion von $G'[S]$ für eine Teilmenge $S \subseteq X$ wegen jedes Element W der Adjazenzliste $O(tw^2)$ Zeit benötigt wird.

Für jedes Element W dieser Liste wird wegen der Gleichung (3.10) genau eine Zeile, nämlich $path_W(Q)$ mit $Q := W \setminus (X \setminus S)$ aus der Tabelle $path_W$ abgelesen. Die Berechnung von Q kann analog zur Überprüfung der Bedingung (3.4) in $O(tw^2)$ Zeit durchgeführt werden. Der Hashwert des Schlüssels Q wird in $O(tw)$ Zeit berechnet. Die Zeile $path_W(Q)$ kann somit in $O(tw^2 + tw)$ Zeit gefunden werden. Auch $S \cap (W \cap X)$ wird analog zur Überprüfung der Bedingung (3.4) in $O(tw^2)$ Zeit berechnet. Danach kann für jedes Knotenpaar $u, v \in S \cap (W \cap X)$ in konstanter Zeit der Vergleich $d(uvWQ) < c'_{uv}$ durchgeführt und im Erfolgsfall c'_{uv} und $p(uv)$ aktualisiert werden ($d(uvWQ)$ ist eine Zelle $d[u][v]$ eines zweidimensionalen Arrays in Zeile $path_W(Q)$). Wegen $|S \cap (W \cap X)| \leq (tw + 1)$ werden höchstens $(tw + 1)^2$ Vergleiche und $2 \cdot (tw + 1)^2$ Aktualisierungen durchgeführt. Das kostet $O(tw^2)$ Zeit. Die Konstruktion von $G'[S]$ benötigt wegen W somit $O(tw^2 + tw) + O(tw^2) + O(tw^2) = O(tw^2)$ Zeit.

Für alle $O(2^{tw})$ Teilmengen $S \subseteq X$ wird die Konstruktion von $G'[S]$ wegen ein Element W der Adjazenzliste von X insgesamt $O(2^{tw} \cdot tw^2)$ Zeit benötigen. Man beachte, dass die Adjazenzliste eines Baumknotens X nur von X aus durchlaufen wird. Der Baum J hat $|V(J)| = O(|V|)$ Baumknoten, daher auch $|E(J)| = O(|V|)$ Baumkanten. Alle Adjazenzlisten können dann in $O(|V(J)| + |E(J)|) = O(|V|)$ Zeit durchlaufen werden. Da für jedes Element der Adjazenzlisten $O\left(2^{tw} \cdot tw^2\right)$ Zeit benötigt wird, kostet die Konstruktion des Hilfsgraphen $G'[S]$ für alle Baumknoten X und für alle Teilmengen $S \subseteq X$ insgesamt $O\left(|V| \cdot 2^{tw} \cdot tw^2\right)$ Zeit.

In jeder Zeile $path_X(S)$ mit $S \subseteq X$ und $|X| \leq tw + 1$ werden für jedes Knotenpaar $u, v \in S$ der kürzeste (u, v)-Pfad im Hilfsgraphen $G'[S]$ beispielsweise mit Dijkstra in Zeit $O(|S|^3) = O\left(tw^3\right)$ berechnet. Für alle Tabellen und alle Zeilen wird somit $O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ Zeit benötigt. Weil außerdem alle Hashtabellen in erwarteter Zeit $O(|V| \cdot 2^{tw} \cdot tw)$ und alle Hilfsgraphen in Zeit $O\left(|V| \cdot 2^{tw} \cdot tw^2\right)$ erstellt werden, folgt die Behauptung. □

3.4.3 Traceback für relevante Pfade

Irgendwann hat der Algorithmus für relevante Pfade alle Tabellen berechnet. Die Kanten eines relevanten Pfads P sind eventuell in mehrere Tabellen $path_{X_1}, \cdots, path_{X_l}$ für die Baumknoten $X_1, \cdots, X_l$ gestreut. Wie können wir die Kanten von P aus dieser Tabellen ablesen? Dazu verwenden wir das Traceback.

Traceback für einen kürzesten relevanten Pfad (Traceback-1)

Gegeben seien ein Baumknoten X mit Elter Y, ein $S \subseteq X$ und ein Knotenpaar $u, v \in S$. Wie können wir die Kantenmenge E' eines in $G[V_X^Y \setminus (X \setminus S)]$ kürzesten relevanten (u, v)-Pfads mit Kosten $d < +\infty$ aus der Tabellen ablesen?

Algorithmus Traceback-1. Zuerst erstellen wir eine leere Kantenmenge E'. Der Algorithmus für relevante Pfade hat anstelle des in $G[V_X^Y \setminus (X \setminus S)]$ kürzesten relevanten (u, v)-Pfads einen im Hilfsgraphen $G'[S]$ kürzesten Pfad $uvXS$ mit Kosten d gespeichert. Wir bearbeiten $uvXS$ wie folgt. Sei c'_e die Kosten der Kante e des Hilfsgraphen. Nach Lemma 3.4.3 gibt es einen in $G[V_X^Y \setminus (X \setminus S)]$ kürzesten (u, v)-Pfad P_{uv} mit Kosten d, so dass für jede Kante wz von $uvXS$ gilt:

- Ist wz eine echte Kante, dann ist wz auch eine Kante von P_{uv}. Daher fügen wir wz in E' ein.

- Ist wz eine künstliche Kante mit Kosten c'_{wz}, so enthält P_{uv} anstelle dieser Kante einen Teilpfad, der ein (w, z)-Pfad mit Kosten c'_{wz} ist. Wegen der Gleichung (3.10) gibt es ein Kind W von X, so dass $d(wzWQ) = c'_{wz}$ für $Q = W \setminus (X \setminus S)$ ist. Wegen der Konstruktion zeigt der Zeiger $p(uv)$ auf diesen Pfad: $p(wz) = wzWQ$.

  Da es den Pfad $wzWQ$ mit $d(wzWQ) = c'_{wz}$ gibt, folgt aus Lemma 3.4.3, dass es in $G[V_W^X \setminus (W \setminus Q)]$ einen (w, z)-Pfad P_{wz} mit Kosten c'_{wz} gibt. Um die Kantenmenge von P_{wz} zu finden kann man daher den Pfad $wzWQ$ analog zu $uvXS$ bearbeiten. Wir setzen fort, indem wir $wzWQ$ rekursiv analog zu $uvXS$ bearbeiten.

Die Korrektheit folgt aus der Korrektheit des Algorithmus für relevante Pfade.

Traceback für k kürzeste relevante Pfade (Traceback-k)

Angenommen, wir suchen die Kantenmenge E' der k Pfade, die von dem Algorithmus für relevante Pfade berechnet sind. Man kann natürlich mit dem Algorithmus Traceback-1 einen Pfad nach dem anderen bearbeiten und ihre Kanten sammeln. Wenn beispielsweise $k = O(|V|)$ ist und die Kanten jedes Pfads in $O(|V|)$ Tabellen gestreut sind, würden wir dafür $\Omega(|V|^2)$ Zeit benötigen. Das wäre jedoch ineffizient.

Der Algorithmus für relevante Pfade berechnet und speichert insgesamt höchstens $N := |V| \cdot 2^{tw+1} \cdot (tw+1)^2$ Pfade so dass jeder Pfad höchstens tw Kanten hat. Der Grund dafür ist Folgendes: Für jeden Baumknoten $X \neq R$ wird eine Tabelle $path_X$ mit höchstens 2^{tw+1} Zeilen erstellt. In jeder Zeile $path_X(S)$ mit $S \subseteq X$ und $|X| \leq tw+1$ werden für jedes Knotenpaar $u, v \in S$ der kürzeste (u,v)-Pfad im Hilfsgraphen $G'[S]$ berechnet und gespeichert. Wegen $|S| \leq |X| \leq tw+1$ gibt es höchstens $(tw+1)^2$ Knotenpaare in S. Jeder kürzeste (u,v)-Pfad in $G'[S]$ kann wegen $|S| \leq tw+1$ höchstens tw Kanten haben. Nach Bemerkung 3.1.3 hat J höchstens $|V|+1$ Baumknoten. Daraus folgt die Behauptung.

Wenn wir jede von N Pfaden höchstens einmal bearbeiten würden, dann wäre die Laufzeit $O(k + N \cdot tw)$. Diese Laufzeit ist linear in $|V|$, wenn tw konstant und $k = O(|V|)$ ist. Wir müssen also jeden von N Pfaden höchstens einmal bearbeiten. Ein Pfad P wird nur dann mehrmals bearbeiten, wenn er ein Teilpfad von mehreren Pfaden $P_1, P_2, \cdots$ ist. Das werden wir verhindern, indem wir P nur erstmalig, beispielsweise bei der Berechnung von P_1 berechnen. Dazu sei bemerkt, dass wir von einem Pfad P nur seine Kanten brauchen. Das heißt, nachdem wir einmal alle seine Kanten während der Bearbeitung des Pfads P_1 gefunden haben, können wir P als *"berechnet"* markieren, damit seine Kanten während der Bearbeitung von P_2 nicht ein zweites Mal gesucht werden.

Algorithmus Traceback-k. Wir erstellen eine leere Kantenmenge E'. Für jede von k gesuchten Pfaden, gehen wir wie folgt vor: Ist der Pfad ein (u,v)-Pfad, der in Zeile $path_X(S)$ verwaltet wird, wird der Pfad $uvXS$ bearbeitet und die gefundene echte Kanten werden in die Kantenmenge E' eingefügt. Dabei geht Traceback-k bis auf eine Ausnahme wie Traceback-1 vor. Initial gibt es keinen Pfad mit Markierung *"berechnet"*. Für jeden zu bearbeitenden Pfad wird die folgende *Markierungsregel* angewendet.

- Ein Pfad wird bearbeitet, wenn er noch nicht als *"berechnet"* markiert ist.
- Sobald die Bearbeitung eines Pfads angefangen hat, wird er als *"berechnet"* markiert.

Abschließend sei bemerkt, wie Traceback-1 ist auch Traceback-k rekursiv. Das heißt, während der Bearbeitung eines Pfads P kann das Traceback-k für die Teilpfade $P_1, P_2, \cdots$ von P aufgerufen werden. Die Pfade $P_1, P_2, \cdots$ werden dann auch der Markierungsregel unterzogen.

3.4.5 Bemerkung. (Markierung) Um die Pfade einer Zeile $path_X(S)$ zu markieren, erstellen wir ein zweidimensionales Array b in dieser Zeile: Ist ein Pfad $uvXS$ als *"berechnet"* markiert, so ist $b[u][v] = 1$, sonst ist $b[u][v] = 0$. Mit diesem Array kann man einen Pfad in konstanter Zeit markieren, aber auch abfragen, ob er markiert ist.

3.4.6 Bemerkung. (Mengenoperation) Um die Kantenmenge E' effizient zu verwalten, erstellen wir einmalig eine Variable $isSelected(e)$ für jede Kante $e \in E$. Diese Variable

wird als *statisches perfektes Hashing* [7] realisiert, wobei jede Kante $e \in E$ ein Schlüssel der Hashtabelle *isSelected* ist. Für s Schlüssel erstellt statisches perfektes Hashing eine Tabelle in erwarteter Zeit $O(s)$, so dass anschließend jeder Schlüssel in konstanter Zeit gefunden wird. Wir dürfen statisches perfektes Hashing benutzen, da alle Schlüssel von vornerein bekannt sind und jeder Schlüssel nur einmal am Anfang eingefügt und dann nie gelöscht wird. Die Hashtabelle wird in erwarteter Zeit $O(|E|)$ erstellt. Für jede Kante $e \in E$ wird initial in $O(1)$ Zeit $isSelected(e) := 0$ gesetzt. Für alle Kanten kostet das $O(|E|)$ Zeit. Wird eine Kante e in die Menge E' eingefügt, so aktualisieren wir $isSelected(e) := 1$. Jede Kante e kann somit in $O(1)$ Zeit in die Menge E' eingefügt werden. Will man die Kantenmenge E' ausgeben, so durchläuft man alle Kanten $e \in E$ und gibt alle Kanten e mit $isSelected(e) = 1$ aus. Für alle Kanten kostet das $O(|E|)$ Zeit.

3.4.7 Lemma. *Traceback-k findet die Kantenmenge E' in Zeit $F+O\left(k + |V| \cdot 2^{tw} \cdot tw^3\right)$, wobei F eine Zufallsvariable mit Erwartungswert $O(|V| \cdot tw)$ ist.*

Beweis. Zuerst zeigen wir die Aussage über die Laufzeit. Wir haben oben erwähnt, dass der Algorithmus für relevante Pfade in allen Tabellen insgesamt höchstens $N := |V| \cdot 2^{tw+1} \cdot (tw+1)^2$ verschiedene relevante Pfade $P_1, \cdots, P_N$ speichert, die jeweils höchstens tw Kanten haben.

Da der Algorithmus Traceback-k rekursiv ist, betrachtet er eventuell mehr als k Pfade. Dies passiert genau dann, wenn wir einen Pfad $P \in \{P_1, \cdots, P_N\}$ betrachten und feststellen, dass er noch nicht als *"berechnet"* markiert ist und künstliche Kanten enthält. Da jede künstliche Kante einen Teilpfad vertritt, muss auch dieser Teilpfad betrachtet werden. Für jede künstliche Kante von P wird daher ein Pfad betrachtet. Da jeder Pfad $P \in \{P_1, \cdots, P_N\}$ höchstens tw Kanten hat, kann wegen P höchstens tw Pfade betrachtet werden, aber nur wenn P nicht als *"berechnet"* markiert ist. Im Laufe des Algorithmus kann P nur einmal ohne die Markierung *"berechnet"* betrachtet werden. Denn sobald P einmal betrachtet wurde, wird er als *"berechnet"* markiert. Im Laufe des Algorithmus kann daher wegen einen Pfad $P \in \{P_1, \cdots, P_N\}$ nur einmal und zwar höchstens tw Pfade betrachtet werden. Für alle N verschiedene Pfade $\{P_1, \cdots, P_N\}$ werden dann höchstens $N \cdot tw$ Pfade betrachtet. Da die k Pfade zwangsläufig betrachtet werden sollen, werden im Laufe des Algorithmus höchstens $L := k + N \cdot tw$ Pfade betrachtet.

Wie lange dauert die Bearbeitung von L Pfaden? Die Abfrage, ob P markiert ist, kann nach Bemerkung 3.4.5 in $O(1)$ Zeit beantwortet werden. Ist ein Pfad P schon als *"berechnet"* markiert, so wird er in $O(1)$ Zeit verworfen. Sonst werden seine $O(tw)$ Kanten betrachtet. Das kostet $O(tw)$ Zeit, da jede echte Kante in konstanter Zeit in E' eingefügt (Bemerkung 3.4.6) und zu jeder künstlichen Kante wz entsprechender Pfad mit Hilfe des Zeigers $p(wz)$ in konstanter Zeit zugegriffen werden kann. Im Laufe des Algorithmus kann jeder Pfad $P \in \{P_1, \cdots, P_N\}$ nur einmal ohne der Markierung *"berechnet"* betrachtet werden und $O(tw)$ Zeit kosten, danach wird immer in $O(1)$ Zeit verworfen. Da es N verschiedene Pfade

$P_1, \cdots, P_N$ gibt, kann es höchstens N Pfade einmalig $O(tw)$ Zeit kosten, die restliche $L-N$ Pfade werden jeweils in $O(1)$ Zeit verworfen. Die Bearbeitung von L Pfaden kostet dann $O(N \cdot tw + (L-N) \cdot 1) = O(L)$ Zeit.

Wir fassen zusammen: Traceback-k betrachtet $O(L)$ nicht unbedingt verschiedene Pfade und bearbeitet sie insgesamt in $O(L)$ Zeit. Für die Ausgabe von E' wird auch einmalig $O(|E|)$ Zeit benötigt. Außerdem wird für die Verwaltung von E' (Bemerkung 3.4.6) einmalig eine Hashtabelle in erwarteter Zeit $O(|E|)$ erstellt. Nach Lemma 2.3.3 ist $|E| = O(|V| \cdot tw)$. Die Laufzeit beträgt somit insgesamt $F + O(L + |V| \cdot tw)$ Zeit, wobei F eine Zufallsvariable mit Erwartungswert $\mathbf{E}(F) = O(|V| \cdot tw)$ ist, was zu zeigen war.

Als Nächstes zeigen wir die Korrektheit. Da wir die Kantenmenge E' von k Pfaden suchen, würde die naive Idee den Algorithmus Traceback-1 k Mal aufrufen. Das wäre korrekt, da Traceback-1 korrekt ist und die mehrfach gefundene Kanten in Kantenmenge E' vereinigt werden. Der einzige Unterschied zwischen dem Traceback-k und der naive Idee ist, dass Traceback-k einen Pfad P nur einmal bearbeitet, das heißt, seine Kanten $E(P)$ höchstens einmal berechnet. Das reicht völlig aus, da die berechnete Kanten am Ende in Kantenmenge E' vereinigt werden. Daher ist auch das Traceback-k korrekt. □

3.5 Dynamisches Programm für allgemeine Graphen

Nachdem wir die benötigte Werkzeuge gesammelt haben, können wir den Algorithmus für das Steinerbaumproblem in allgemeinen Graphen formulieren. In Unterkapitel 3.4 haben wir überlegt, warum der Spezialfall-Algorithmus (Unterkapitel 3.3) in allgemeinen Graphen nicht funktioniert. Der Grund war, dass die Teilgraphen, die vom Spezialfall-Algorithmus betrachtet werden, zusammenhängend sein müssen, was in allgemeinen Graphen nicht der Fall ist. Grob formuliert macht der Algorithmus für allgemeine Graphen zuerst die zu betrachtenden Teilgraphen zusammenhängend und wendet dann den Spezialfall-Algorithmus an.

3.5.1 Algorithmus für das Steinerbaumproblem

Wegen der Bemerkung 3.4.2 dürfen wir annehmen, dass G keine Kante mit unendlichen Kosten enthält (gegebenenfalls entfernen wir sie).

Sei $(J, \mathcal{X})$ eine Baumzerlegung mit Baumweite tw des Graphen G, wobei J gemäß der Idee 3.1.1 mit R gewurzelt ist. Zuerst wird der *Algorithmus für relevante Pfade* (Abschnitt 3.4.2) einmalig durchgeführt und seine Ergebnisse in Tabelle $path_X$ für jeden Baumknoten $X \neq R$ gespeichert.

Als Nächstes wird die Konstruktion von H_X^Y für jeden Baumknoten $X \neq R$ mit Elter Y gestartet. Bis auf eine Ausnahme geht das ähnlich zu dem Spezialfall-Algorithmus (Unterkapitel 3.3). In Phase 1 der Bearbeitung eines Baumknotens X mit Elter Y konstruiert der Spezialfall-Algorithmus den Teilgraphen H_X^Y, der aus X nur die Knotenteilmenge

$S \subseteq X$ enthalten soll, im Teilgraphen $G[S]$. Hier dagegen findet die Konstruktion in einem vollständigen Hilfsgraphen $G'[S]$ statt, dessen jede Kante uv mit der Länge des kürzesten relevanten (u, v)-Pfads im Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$ gewichtet sind: $c'_{uv} = d(uvXS)$.

Da der Hilfsgraph G' sowohl echte als auch künstliche Kanten hat, wird auch H echte und künstliche Kanten haben. Hat der Teilgraph H endliche Kosten, so vertreten seine künstliche Kanten Pfade in G. Diese Pfade können in H Kreise erzeugen, sogar gemeinsame Kanten haben. Die Kosten der gemeinsamen Kanten werden dann mehrfach gezählt. Da die Kantenkosten positiv sind, macht es Sinn, am Ende einen Spannbaum auf H zu berechnen. Dazu finden wir zuerst alle echte Kanten von H.

Wegen der Konstruktion enthält tab_R nur eine Zeile. Wir starten bei dieser Zeile und durchlaufen H, indem wir die Zeigerliste verfolgen, wie es in Unterkapitel 3.2 beschrieben ist. Dabei sammeln wir alle echte und künstliche Kanten von H aus der besuchten Zeilen. Bezüglich der echten Kanten gibt es nichts zu tun. Die künstliche Kanten müssen noch durch die entsprechende Pfade ersetzt werden.

Wegen der Konstruktion vertritt jede künstliche Kante einen von dem Algorithmus für relevante Pfade berechneten Pfad. Angenommen, $E(H)$ hat k künstliche Kanten. Dann müssen wir k Pfade, genauer gesagt ihre Kanten E' berechnen. Wir entfernen diese k künstliche Kanten aus der Kantenmenge $E(H)$ und berechnen die entsprechende Pfade mit dem Traceback-k. Das Traceback-k gibt die Kantenmenge E' der berechneten Pfade aus. Wir fügen diese Kanten in $E(H)$ ein: $E(H) := E(H) \cup E'$.

3.5.1 Lemma. *Nachdem der Algorithmus für das Steinerbaumproblem alle Tabellen berechnet hat, können alle echte Kanten von H in Zeit $F + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ aus den Tabellen abgelesen werden, wobei F eine Zufallsvariable mit Erwartungswert $O(|V| \cdot tw)$ ist.*

Beweis. Für die Korrektheit muss nur gezeigt werden, dass nachdem alle echten und künstlichen Kanten von H aus den Tabellen abgelesen wurden, die künstlichen Kanten durch den entsprechenden relevanten Pfade korrekt ersetzt wurden. Da wir dazu den Algorithmus Traceback-k benutzt haben, folgt die Behauptung aus der Korrektheit des Algorithmus Traceback-k.

Als Nächstes zeigen wir die Aussage über die Laufzeit. Zuerst werden alle echten und künstlichen Kanten von H gesammelt. Dazu wird aus jeder Tabelle tab_X eine Zeile $tab_X(S)$ besucht. Sei Y der Elter von X. In der Zeile $tab_X(S)$ stehen $O(tw^2)$ Kanten von $H_X^Y \cap G[X]$ mit $V(H_X^Y) \cap X = S$. Denn $H_X^Y \cap G[X] = H_X^Y[S]$ ist ein Teilgraph eines vollständigen Graphen $G'[S]$ mit $|S| \leq tw + 1$ Knoten. Aus einer Zeile $tab_X(S)$ kann daher höchstens $O(tw^2)$ Kanten gesammelt werden. Man beachte, dass aus einer Tabelle tab_X nur eine Zeile $tab_X(S)$ abgelesen wird, da jede Zeile einer anderen Lösung entspricht. Daher kann H in jeder Tabelle tab_X höchstens $O(tw^2)$ Kanten haben. Der Zugriff auf einen Baumknoten dauert somit $O(tw^2)$ Zeit.

Nachdem wir alle Kanten aus einer Zeile $tab_X(S)$ gesammelt haben, verfolgen wir jeden Zeiger aus der Zeigerliste dieser Zeile. Wegen der Konstruktion enthält diese Liste für jedes Kind W von X höchstens einen Zeiger, der genau auf eine tab_W-Zeile zeigt. Im schlimmsten Fall enthält die Zeigerliste einer Zeile $tab_X(S)$ genau einen Zeiger für jedes Element der Adjazenzliste von X, weshalb die Adjazenzliste völlig durchlaufen werden muss.

Der Baum J hat $|V(J)| = O(|V|)$ Baumknoten (Bemerkung 3.1.3), daher auch $|E(J)| = O(|V|)$ Baumkanten. Da der Zugriff auf ein Element der Adjazenzlisten, also auf einen Baumknoten $O(tw^2)$ Zeit dauert (siehe oben: aus jeder Tabelle müssen $O(tw^2)$ Kanten von H gesammelt werden), können alle Adjazenzlisten in $O\left(|V(J)| + |E(J)|\right) \cdot O(tw^2) = O(|V| \cdot tw^2)$ Zeit durchlaufen und alle Kanten von H gesammelt werden.

H hat somit $k := O(|V| \cdot tw^2)$ Kanten, daher auch höchstens k künstliche Kanten. Nach Lemma 3.4.7 können zu diesen k künstlichen Kanten entsprechende k Pfade mit dem Algorithmus Traceback-k in Zeit $F + O\left(k + |V| \cdot 2^{tw} \cdot tw^3\right) = F + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ berechnet werden, wobei F eine Zufallsvariable mit Erwartungswert $\mathbf{E}(F) = O(|V| \cdot tw)$ ist. Daraus folgt die Behauptung. □

3.6 Analyse

Da der Korrektheitsbeweis viele Details enthält, werden wir ihn nicht in einem einzigen Lemma zeigen. Das Ziel ist zu zeigen, dass der Algorithmus einen zusammenhängenden Teilgraphen H von G berechnet, der alle Terminalknoten enthält. Dazu werden wir wie folgt vorgehen. Sei R die Wurzel des Baums J. In Lemma 3.6.2 werden wir zeigen, dass der Algorithmus für jeden Baumknoten $X \neq R$ mit Elter Y einen Teilgraphen H_X^Y mit endlichen Kosten konstruiert, der die Bedingung **H*** erfüllt. Dazu werden wir das Hilfslemma 3.6.1 benötigen, das zeigt, dass H_X^Y jeden Terminalknoten aus dem Teilgraphen G_X^Y enthält. Als Nächstes werden wir das Lemma 3.6.3 betrachten, das für die schnelle Verkettung benötigt wird. Anschließend werden wir in Theorem 3.6.4 die Korrektheit und die Laufzeit nachweisen.

3.6.1 Lemma. *Sei $X \neq R$ ein Baumknoten mit Elter Y und S eine Knotenteilmenge von X. Hängt im Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$ jeder Terminalknoten aus V_X^Y und jeder Knoten aus S mit einem Knoten aus $X \cap Y$ zusammen, so konstruiert der Algorithmus einen Teilgraphen H_X^Y von G_X^Y mit $c(H_X^Y) < +\infty$ und $V(H_X^Y) \cap X = S$, so dass:*

i) *jeder Knoten aus S und jeder Terminalknoten aus V_X^Y in H_X^Y mit einem Knoten aus $X \cap Y$ verbunden ist.*

ii) *Sei c in Zeile $tab_X(S)$ berechnete Kosten. Dann sind die Kosten des Teilgraphen H_X^Y höchstens c.*

Beweis. Das zeigen wir mit vollständiger Induktion über die Baumknoten.

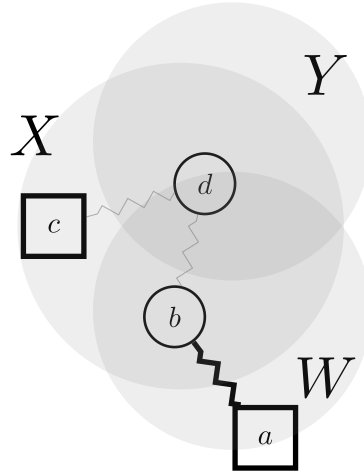


Abbildung 3.15: Durchschnittsmengen von X mit dem Elter Y und dem Kind W

IA: *i)* Sei X ein Blatt. Zuerst müssen wir zeigen, dass S die Bedingungen (3.1) und (3.4) erfüllt, sonst würde der Algorithmus die entsprechende Zeile $tab_X(S)$ ohne zu bearbeiten mit $+\infty$ bewerten. Da X ein Blatt ist, ist $V_X^Y = X$ und $V_X^Y \setminus (X \setminus S) = S$. Daher hängt wegen der Voraussetzung des Lemmas jeder Terminalknoten aus X in $G[S]$ mit einem $X \cap Y$ Knoten zusammen. Das heißt Folgendes: Erstens ist jeder Terminalknoten aus X auch in S und daher ist die Bedingung (3.1) erfüllt. Zweitens hat S mindestens einen Knoten aus $X \cap Y$ und daher ist auch die Bedingung (3.4) erfüllt. Es bleibt nur noch zu zeigen, dass H_X^Y mit $V(H_X^Y) \cap X = S$ jeden $S \setminus (X \cap Y)$ Knoten über einen Pfad mit endlichen Kosten mit einem $X \cap Y$ Knoten verbindet. Algorithmus startet Prim in Hilfsgraphen $G'[S]$, der genau dann einen (u, v)-Pfad mit Kosten c enthält, wenn auch $G[V_X^Y \setminus (X \setminus S)] = G[S]$ einen (u, v)-Pfad mit Kosten c enthält. Wegen der Annahme hängt jeder Knoten aus S in $G[S]$ mit einem $X \cap Y$ Knoten zusammen, somit auch in $G'[S]$. Daher kann Prim in $G'[S]$ jeden $S \setminus (X \cap Y)$ Knoten mit einem $X \cap Y$ Knoten mit endlichen Kosten verbinden.

ii) H_X^Y besteht nur aus den Kanten, die von Prim aus $G'[S]$ gewählt wurden. Die Summe c der Kosten dieser Kanten werden in Zeile $tab_X(S)$ gespeichert. Es gilt sogar $c(H_X^Y) = c$.

IS: *i)* Falls X ein Blatt ist, kann die Korrektheit analog zu **IA** gezeigt werden. Angenommen, das Gegenteil wäre der Fall. Der Teilgraph $G[V_X^Y \setminus (X \setminus S)]$ besteht aus überlappenden Teilgraphen $G[S]$ und $G[V_W^X \setminus (X \setminus S)]$ für jedes Kind W von X. Wir müssen daher für jeden von diesen Teilgraphen zeigen, dass, wenn er einen Terminalknoten hat, hängt dieser Terminalknoten in H_X^Y mit einem $X \cap Y$ Knoten zusammen. Falls ein Teilgraph $G[V_W^X]$ keinen Terminalknoten hat, dann gibt es bezüglich diesem Teilgraphen nichts zu zeigen. Wir nehmen daher an, dass jeder von diesen Teilgraphen mindestens einen Terminalknoten hat.

Wegen **tw3** ist $S \cap (W \cap X)$ die überlappende Knotenmenge der Teilgraphen $G[S]$ und $G[V_W^X \setminus (X \setminus S)]$. Jeder Terminalknoten $t \in V_W^X$ (beispielsweise a in Abbildung 3.15) ist in $G[V_W^X \setminus (X \setminus S)]$ mit einem Knoten aus $S \cap (W \cap X)$ (beispielsweise b in Abbildung 3.15) zusammengehängt, sonst wäre t in $G[V_X^Y \setminus (X \setminus S)]$ mit keinem Knoten aus $X \cap Y$ zusammenhängend, im Widerspruch zur Annahme. Das heißt $S \cap (W \cap X)$ hat mindestens einen Knoten:

$$S \cap (W \cap X) \neq \emptyset$$

Dann gibt es eine Teilmenge $Q \subseteq W$ mit

$$Q \cap (W \cap X) = S \cap (W \cap X) \neq \emptyset, \tag{3.11}$$

so dass jeder Knoten aus Q und jeder Terminalknoten aus V_W^X in $G[V_W^X \setminus (W \setminus Q)]$ mit einem Knoten aus $W \cap X$ zusammenhängt. Der Grund dafür ist Folgendes. Seien $t_1, \cdots, t_j$ die Terminalknoten in V_W^X. Jeder Terminalknoten t_i mit $1 \leq i \leq j$ ist in $G[V_W^X \setminus (X \setminus S)]$ auf einem Pfad P_i mit endlichen Kosten mit einem Knoten aus $S \cap (W \cap X)$ verbunden. Dann erfüllt $Q := W \cap V(P_1 \cup \cdots \cup P_j)$ die obige Bedingung. Dann konstruiert der Algorithmus wegen **IV** für jede Teilmenge $Q \subseteq W$, die die obige Bedingung erfüllt, einen Teilgraphen H_W^X von $G[V_W^X \setminus (W \setminus Q)]$ mit $c(H_W^X) < +\infty$ und $V(H_W^X) \cap W = Q$, der jeden Terminalknoten aus V_W^X mit einem Knoten aus $W \cap X$ verbindet. Dieser Teilgraph wird in Zeile $tab_W(Q)$ gespeichert. Wir zeigen jetzt, dass falls der Algorithmus die Zeile $tab_X(S)$ mit einer Zeile $tab_W(Q)$ verkettet, $Q \subseteq W$ dann die obige Bedingung somit auch die Gleichung (3.11) erfüllt. Damit der Algorithmus die Zeile $tab_X(S)$ mit einer Zeile $tab_W(Q)$ verkettet, müssen die Bedingungen (3.5) und (3.9) erfüllt sein. Setzt man $Y := X$, $X := W$, $P := S$, $S := Q$ in beiden Bedingungen ein und beachtet die Annahme $V_W^X \cap T \neq \emptyset$, so erfüllt die Gleichung (3.11) die Beiden.

Es bleibt nur noch zu zeigen, dass H_X^Y mit $V(H_X^Y) \cap X = S$ jeden $S \setminus (X \cap Y)$ Knoten mit einem $X \cap Y$ Knoten über einen Pfad mit endlichen Kosten verbindet. Analog zu **IA** kann man zeigen, dass S die Bedingungen (3.1) und (3.4) erfüllt, daher nicht ignoriert wird. (Erinnerung: Falls die beide Bedingungen nicht erfüllt sind, wird die entsprechende Teillösung direkt mit $+\infty$ bewertet und nicht weiter betrachtet.) Für die Konstruktion von H_X^Y sind nur die (u, v)-Pfade mit $u, v \in S$ nötig, die völlig in $G[V_X^Y \setminus (X \setminus S)]$ liegen. Der Algorithmus startet Prim in Hilfsgraphen $G'[S]$, der genau dann einen (u, v)-Pfad mit Kosten c enthält, wenn auch $G[V_X^Y \setminus (X \setminus S)]$ einen (u, v)-Pfad mit Kosten c enthält. Wegen der Voraussetzung des Lemmas hängt jeder Knoten aus S in $G[V_X^Y \setminus (X \setminus S)]$ mit einem $X \cap Y$ Knoten zusammen, somit auch in $G'[S]$. Daher kann Prim in $G'[S]$ jeden $S \setminus (X \cap Y)$ Knoten mit einem $X \cap Y$ Knoten mit endlichen Kosten verbinden.

ii) Seien $W_1, \cdots, W_l$ die Kinder von X und

$$Q_i := \text{argmin}_Q\{tab_{W_i}(Q) \colon Q \subseteq W_i,\ Q \cap (W_i \cap X) = S \cap (W_i \cap X)\}$$

für $1 \leq i \leq l$. Die Kosten c der Zeile $tab_X(S)$ setzt sich zusammen aus den Kantenkosten, die von Prim aus $G'[S]$ gewählt wurden und $\sum_{i=1}^{l} tab_{W_i}(Q_i)$. Wegen Induktionsvoraussetzung ist $tab_{W_i}(Q_i)$ korrekt berechnet, daher ist auch $tab_X(S)$ richtig berechnet. Der Grund, warum H_X^Y höchstens c und nicht genau c kostet, ist, dass eine $G'[S]$ Kante eventuell einen Pfad vertreten kann, der eine Kante aus H_W^X für ein Kind W von X enthält. In diesem Fall sind die Kosten dieser Kante in c zweifach enthalten. □

3.6.2 Lemma. *Für jeden Baumknoten $X \neq R$ mit Elter Y erzeugt der Algorithmus einen Teilgraphen H_X^Y mit $c(H_X^Y) < +\infty$ von G_X^Y, der die Bedingung $\boldsymbol{H^*}$ erfüllt.*

Beweis. Da G zusammenhängend ist, gibt es eine Teilmenge $S \subseteq X$, die die Bedingungen von Lemma 3.6.1 erfüllt. Im Folgenden betrachten wir den Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = S$. Wir müssen Folgendes zeigen:

i) H_X^Y enthält jeden Terminalknoten aus V_X^Y in eine von seiner Komponenten.

ii) Jede Komponente von H_X^Y hat mindestens einen Knoten aus $X \cap Y$.

i) Das haben wir in Lemma 3.6.1 gezeigt.
ii) Zum Widerspruchsbeweis nehmen wir an, dass H_X^Y eine Komponente K hat, die keinen Knoten aus $X \cap Y$ enthält. Wegen der Phase 1 der Bearbeitung von X hängt in H_X^Y jeder $S \backslash (X \cap Y)$ Knoten mit einem Knoten aus $X \cap Y$ zusammen. Dann enthält K keinen Knoten aus S. Da H_X^Y auch keinen Knoten aus $X \setminus S$ hat, hat K daher auch keinen Knoten aus X. Die Komponente von H_X^Y liegen in G_X^Y. Dann gibt es in J_X^Y einen Baumknoten X' mit Elter Y', so dass K einen Knoten aus X' hat, aber keinen aus Y'. Da K in H_X^Y liegt, enthält H_X^Y eine Knotenteilmenge $S' \subseteq X'$, so dass K einen Knoten $v \in S'$ hat. v ist also in $H_X^Y[S']$. Da S' einen Knoten hat, ist sie nicht leer. Dann ist wegen der Bedingung (3.4) auch $S' \cap (X' \cap Y')$ nicht leer. Da K keinen Knoten aus Y' hat, ist v in $S' \setminus (X' \cap Y')$, also in $H_X^Y[S' \setminus (X' \cap Y')]$. Dann hat die Phase 1 von X' den Knoten v in $H_X^Y[S']$ mit einem Knoten $w \in X' \cap Y'$ zusammengehängt. Dann enthält K auch den Knoten $w \in Y'$. Widerspruch! □

Wir haben in Abschnitt 3.3.2 erwähnt, dass um die schnelle Verkettung benutzen zu können, eine Eigenschaft der Phase 1 benötigt wird. Wir werden zeigen, dass diese Bedingung erfüllt ist, wenn die künstliche Kanten die relevante Pfade vertreten. Ferner sei bemerkt, dass diese Bedingung auch bei dem Spezialfall-Algorithmus (Unterkapitel 3.3) erfüllt ist. Da der Spezialfall-Algorithmus auch nur relevante Pfade benutzt.

3.6.3 Lemma. *Ist Z ein noch nicht bearbeiteter Baumknoten, so ist keine Teilmenge des Bags Z in einer bisher konstruierten Teillösung zusammengehängt.*

Beweis. Zum Widerspruchsbeweis nehmen wir an, dass Z *"das erste Bag"* im Laufe des Algorithmus ist, das die Behauptung verletzt.

Angenommen, die Teilmenge $\{z_1, z_2\} \subseteq Z$ ist vor Z während der Bearbeitung des Baumknotens X mit Elter Y zusammengehängt. Dann ist Z nicht im Teilbaum J_X^Y. Denn der Algorithmus bearbeitet einen Baumknoten X erst dann, wenn der Teilbaum J_X^Y außer X keinen nicht bearbeiteten Baumknoten hat. Da z_1 und z_2 während der Bearbeitung des Bags X zusammengehängt wurden, liegt z_1 in einem Baumknoten W des Teilbaums J_X^Y. Da Z aber nicht in diesem Teilbaum ist, hat J_X^Y einen Pfad $W, \cdots, X, Y, \cdots, Z$. Weil z_1 sowohl in W als auch in Z ist, ist z_1 wegen der Bedingung **tw3** der Baumzerlegung in jedem Bag dieses Pfads, somit auch in $X \cap Y$. Analoges gilt für z_2. Bisher haben wir Folgendes festgestellt:

- z_1 und z_2 wurden während der Bearbeitung von X zusammengehängt;
- z_1 und z_2 liegen in dem Separator $X \cap Y$;

Es reicht nur noch zu zeigen, dass während der Bearbeitung von X die Knoten aus dem Separator $X \cap Y$ nicht untereinander zusammengehängt werden.

Die Knoten werden nur in Phase 1 zusammengehängt, als der Teilgraph H_X^Y konstruiert wird. Angenommen, H_X^Y enthält aus X nur die Knotenmenge S. Wegen der Annahme *"das erste Bag"* ist jeder Knoten $v \in S$ vor der Bearbeitung von X in verschiedener Komponente $H_X^Y(v)$ von H_X^Y. Der Algorithmus startet Prim und fügt zu einer Komponente $H_X^Y(v)$, die noch keinen $X \cap Y$ Knoten hat, solange eine Kante aus dem Hilfsgraphen $G'[S]$, bis $H_X^Y(v)$ einen Knoten aus $X \cap Y$ enthält.

Sei uw *"die erste $G'[S]$ Kante"* mit $V\left(H_X^Y(u)\right) \cap (X \cap Y) = \emptyset$, die zu einer Komponente mit mehr als einen $X \cap Y$ Knoten geführt hat. Vor dieser Kante hat die Komponente $H_X^Y(v)$ für jedes $v \in S$, darunter auch $H_X^Y(w)$ höchstens einen $X \cap Y$ Knoten. Nach dieser Kante werden $H_X^Y(u)$, $G'[S]$ Kante uw und $H_X^Y(w)$ verschmolzen. Da $H_X^Y(u)$ noch keinen und $H_X^Y(w)$ höchstens einen Knoten aus $X \cap Y$ hat, hat $H_X^Y(u) \cup H_X^Y(w)$ auch höchstens einen $X \cap Y$ Knoten. Die $G'[S]$ Kante uw ist eventuell eine künstliche Kante und vertritt einen Pfad P mit mehr als zwei Knoten. Wir zeigen, dass P außer u und w keinen weiteren Knoten aus X (somit auch aus $X \cap Y$) hat. Bezüglich der $G'[S]$ Kante uw sind zwei Fälle möglich:

$c'_{uw} < +\infty$: Wegen der Konstruktion vertritt diese Kante einen kürzesten (u, w)-Pfad P mit $c'_{uw} = c(P)$ im relevantem Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$. Zum Widerspruchsbeweis nehmen wir an, dass P noch einen Knoten $v \in X$ mit $u \neq v \neq w$ hat. Dann muss v in S und nicht in $X \setminus S$ liegen, sonst wäre P kein relevanter Pfad. Dann hat $G'[S]$ eine Kante uv mit Kosten c'_{uv}. Wir zeigen, dass $c'_{uv} < c'_{uw}$ ist und daher Prim in

$G'[S]$ die Kante uv der Kante uw vorziehen würde.

P hat die Form: $P = u, \cdots, v, \cdots, w$. Da die Kantenkosten positiv sind, ist der Teilpfad $P' := u, \cdots, v$ kürzer als P: $c(P') < c(P)$. Wegen der Konstruktion ist c'_{uv} die Länge des kürzesten (u, v)-Pfads P'' im relevanten Teilgraphen $G[V_X^Y \setminus (X \setminus S)]$: $c'_{uv} = c(P'')$. Dagegen ist P' irgendein, also nicht unbedingt der kürzeste (u, v)-Pfad in diesem Teilgraphen. Daher gilt: $c'_{uv} = c(P'') \leq c(P') < c(P) = c'_{uw}$.

$c'_{uw} = +\infty$: Solch eine Kante vertritt keinen Pfad aus $G[V_X^Y \setminus (X \setminus S)]$, da in Initialisierungsphase alle Kanten mit unendlichen Kosten aus dem Graph entfernt wurden. Daher ist uw eine künstliche $G'[S]$ Kante, die aus S nichts außer den Knoten u und w enthält.

Somit kann nach dieser Kante keine Komponente mit mehr als einen $X \cap Y$ Knoten entstehen, im Widerspruch zur Annahme *"der erste $G'[S]$ Kante"*. Daher ist die Teilmenge $\{z_1, z_2\} \subseteq Z$ während der Bearbeitung von X nicht zusammengehängt, im Widerspruch zur Annahme. Analog zu $\{z_1, z_2\}$ kann man für jedes Knotenpaar $z'_1, z'_2 \in Z$ zeigen, dass sie vor der Bearbeitung von Z nicht zusammengehängt sind. Widerspruch! □

3.6.4 Theorem. *Der Algorithmus für das Steinerbaumproblem berechnet in Zeit $F + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ einen (nicht optimalen) Steinerbaum, wobei F eine Zufallsvariable mit Erwartungswert $\mathbf{E}(F) = O\left(|V| \cdot 2^{tw} \cdot tw\right)$ ist.*

Beweis. Zuerst zeigen wir die Korrektheit: Wegen der Idee 3.1.1 hat die Wurzel R von J genau ein Kind W. Außerdem hat das Bag R genau einen Terminalknoten r. Nach Lemma 3.6.2 erfüllt H_W^R die Bedingung **H***. Das heißt, jede Komponente von H_W^R enthält einen Knoten aus Knotenteilmenge $W \cap R$, die nur aus einem einzigen Knoten r besteht. Daher ist H_W^R ein zusammenhängender Teilgraph. Außerdem enthält er wegen der Bedingung **H*** alle Terminalknoten aus $G_W^R = G$. Somit ist $H := H_W^R$ ein nicht unbedingt optimaler Steinerbaum. Nach der Konstruktion von H_W^R werden nur die mehrfach benutzten Kanten entfernt und die Kreise zerstört. Daher bleibt H weiterhin zusammenhängend und enthält alle Terminalknoten.

Als Nächstes zeigen wir die Aussage über die Laufzeit: In Initialisierungsphase wird der *Algorithmus für relevante Pfade* (Abschnitt 3.4.2) durchgeführt. Nach Lemma 3.4.4 kostet das $F_1 + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ Zeit, wobei F_1 eine Zufallsvariable mit Erwartungswert $\mathbf{E}(F_1) = O\left(|V| \cdot 2^{tw} \cdot tw\right)$ ist. Als Nächstes wird der Algorithmus für den Spezialfall durchgeführt. Nach Lemma 3.6.3 ist die Bedingung der schnellen Verkettung erfüllt, daher kann sie benutzt werden. Nach Lemma 3.3.2 kostet das $F_2 + O\left(|V| \cdot 2^{tw} \cdot tw^2\right)$ Zeit, wobei F_2 eine Zufallsvariable mit Erwartungswert $\mathbf{E}(F_2) = O\left(|V| \cdot 2^{tw} \cdot tw\right)$ ist. Als Nächstes werden alle echten Kanten von H gefunden. Nach Lemma 3.5.1 nimmt das $F_3 + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ Zeit in Anspruch, wobei F_3 eine Zufallsvariable mit Erwartungswert $\mathbf{E}(F_3) = O(|V| \cdot tw)$ ist. Anschließend wird mit Hilfe des *Depth First Search* Algorithmus in $O(|V| + |E|)$ Zeit

ein Spannbaum auf H berechnet. Nach Lemma 2.3.3 gilt $|E| \leq O(|V| \cdot tw)$. Daher können alle Kreise von H in Zeit $O(|V| \cdot tw)$ zerstört werden.

Wir fassen zusammen: Die Laufzeit beträgt insgesamt $F + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ Zeit. wobei $F := F_1 + F_2 + F_3$ eine Zufallsvariable ist. Wegen der Linearität der Erwartungswert (Gleichung (12.3) in [8]) gilt: $\mathbf{E}(F) = \mathbf{E}(F_1 + F_2 + F_3) = \mathbf{E}(F_1) + \mathbf{E}(F_2) + \mathbf{E}(F_3) = O\left(|V| \cdot 2^{tw} \cdot tw\right)$. Daraus folgt die Behauptung. □

3.6.5 Bemerkung. (Laufzeit) Für die praktische Zwecke kann man die randomisierte Teil F der Laufzeit ignorieren und die Laufzeit als $O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ annehmen. Denn die Wahrscheinlichkeit, dass die randomisierte Teil F größer als $|V| \cdot 2^{tw} \cdot tw^3$ ist, ist kleiner als $2^{-|V| \cdot tw^2}$. Dass diese Wahrscheinlichkeit sehr klein ist, kann man sich leicht an einem Zahlenbeispiel klarmachen: Für einen kleinen Graphen mit $|V| = 10$ und $tw = 4$ ist diese Wahrscheinlichkeit kleiner als $2^{-|V| \cdot tw^2} = 2^{-160}$.

Der Grund dafür ist Folgendes: Die randomisierte Teil $F := F_1 + F_2 + F_3$ mit

- $\mathbf{E}(F_1) = O\left(|V| \cdot 2^{tw} \cdot tw\right)$,
- $\mathbf{E}(F_2) = O\left(|V| \cdot 2^{tw} \cdot tw\right)$,
- $\mathbf{E}(F_3) = O(|V| \cdot tw)$.

der Laufzeit entsteht durch die Konstruktion der Hashtabellen, die mit *statischem perfekten Hashing* [7] erstellt werden. Das statisches perfektes Hashing wählt zufällig eine Hashfunktion und erstellt mit Hilfe dieser Funktion eine Hashtabelle. Hat diese Hashtabelle die gewünschte Eigenschaft nicht (ist sie zu groß), wird sie zerstört und der Vorgang solange wiederholt, bis man eine Tabelle mit der gewünschten Größe erstellt hat. Der Punkt ist, dass die Wahrscheinlichkeit dabei, dass die Tabelle die gewünschte Eigenschaft nicht hat, kleiner als $\frac{1}{2}$ ist. Das heißt, die Wahrscheinlichkeit, dass der Vorgang i Mal wiederholt wird, ist kleiner als 2^{-i}.

Beispielsweise betrachten wir F_1. Die Überlegungen gelten analog für F_2, F_3. F_1 entsteht wegen der Konstruktion von $O(|V|)$ Tabellen, wobei die Konstruktion einer Tabelle $O(2^{tw} \cdot tw)$ Zeit benötigt. Wird die Konstruktion einer Tabelle i Mal wiederholt, so wird für diese Wiederholungen $i \cdot O(2^{tw} \cdot tw)$ Zeit verbraucht. Es gilt also:

$$F_1 = O\left(|V| \cdot 2^{tw} \cdot tw\right) + i \cdot O(2^{tw} \cdot tw).$$

Damit F_1 mindestens $O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ groß ist, muss die Anzahl der Wiederholungen $i = |V| \cdot tw^2$ sein. Die Wahrscheinlichkeit dafür ist kleiner als $2^{-|V| \cdot tw^2}$.

Kapitel 4

Evaluierung

4.1 Implementierung

Der Algorithmus wurde in der Programmiersprache Python implementiert. Der Quellcode dieser Implementierung ist auf der beigefügten CD enthalten. Für die Implementierung wurde die Bibliothek SAGE (System for Algebra and Geometry Experimentation) [16] benutzt. SAGE ist ein freies Mathematiksystem, das unter der GNU General Public License (GPL) steht. Die Bibliothek bietet zahlreiche Algorithmen und Datenstrukturen für Graphentheorie an.

4.1.1 Join-Tree Konstruktion

Sei $(J, \mathcal{X})$ eine Baumzerlegung von G. Der Baum J wird in der Literatur oft *Junction-Tree* oder *Join-Tree* genannt. Die Konstruktion von J wurde nach folgendem Algorithmus [11] durchgeführt.

1. Berechne ein Eliminationsschema mit Hilfe der *lexikographischen BFS-Heuristik* [14].
2. Trianguliere G anhand dieses Eliminationsschemas [13] und finde alle maximale Cliquen $\mathcal{X}$. Nach Lemma 2.3.1 gibt es höchstens $|V|$ maximale Cliquen.
3. Konstruiere einen Graphen I, wobei jede Clique $X \in \mathcal{X}$ ein Knoten dieses Graphen ist. Für jedes Cliquenpaar X, Y mit $|X \cap Y| > 0$ enthält I eine Kante XY mit Gewicht $|X \cap Y|$.
4. Berechne einen *maximalen Spannbaum*, also einen Spannbaum mit maximalen Kosten auf I. Dieser Spannbaum ist J.

Für die Implementierung dieser Konstruktion bietet SAGE die Operationen `lex_BFS()` (lexikographische BFS) und `min_spanning_tree()`. Um den maximalen Spannbaum auf I zu berechnen, negieren wir zuerst die Kantengewichte in I und erhalten den Graphen I'. Dann ist ein minimaler Spannbaum J auf I' offensichtlich ein maximaler Spannbaum auf I.

Die SAGE-Funktion `min_spanning_tree()` kann sowohl den Kruskal- als auch den Prim-Algorithmus benutzen. Man beachte, dass diese Algorithmen für beliebige Kostenfunktion $c : E \to \mathbb{R}$ korrekt funktionieren. Es stellt daher kein Problem dar, dass I' Kanten mit negativen Kosten hat.

4.1.1 Bemerkung. (Baumweite tw) Im Folgenden ist mit tw nicht die Baumweite des Graphen, sondern die Baumweite der in Abschnitt 4.1.1 berechneten Baumzerlegung gemeint.

4.1.2 Einige Abweichungen

Es gibt einige Abweichungen zwischen der theoretischen Beschreibung der Heuristik und ihrer Implementierung. Weil wir schon die theoretische Laufzeit der Heuristik festgestellt haben und Python-Programme eine geringe Ausführungsgeschwindigkeit haben, wurde die experimentelle Laufzeitanalyse nicht durchgeführt. Da aber die theoretische Güte der Heuristik nicht nachgewiesen wurde, werden wir die experimentelle Güte der Heuristik untersuchen. Daher wurde bei den Abweichungen darauf geachtet, dass die Güte der Heuristik nicht geändert wird. Die Laufzeit der Implementierung wird dagegen schlechter. Der Grund für die Abweichungen war die Einfachheit der Implementierung.

1. Bei der theoretischen Beschreibung der Heuristik (Abschnitt 3.5.1) wird zuerst der *Algorithmus für relevante Pfade* (Abschnitt 3.4.2) durchgeführt und für alle Baumknoten X die Tabelle $path_X$ berechnet. Erst dann wird die Berechnung der Tabelle tab_X für einen Baumknoten X angefangen. Bei der Implementierung wurde dagegen die Berechnung von $path_X$ und tab_X für jeden Baumknoten X quasi parallel durchgeführt. Das heißt, für den Baumknoten X wurde zuerst $path_X$ und dann tab_X berechnet. Das ist auch korrekt, da für die Berechnung von tab_X nur die Werte aus $path_X$ benötigt werden. Außerdem wurde für jeden Baumknoten X nicht zwei, sondern nur eine Tabelle erstellt, die sowohl die Spalten von tab_X, als auch von $path_X$ enthält.

2. Der Algorithmus für das Steinerbaumproblem in allgemeinen Graphen (Abschnitt 3.5.1) konstruiert zuerst einen Steinerbaum, der eventuell künstliche Kanten enthält. Als Nächstes wird jede künstliche Kante durch den entsprechenden kürzesten relevanten Pfad ersetzt. An dieser Stelle gibt es folgende Abweichung: Gegeben seien ein Baumknoten X mit Elter Y und ein $S \subseteq X$. Sei uv eine künstliche Kante mit $u, v \in S$, die durch einen kürzesten relevanten (u, v)-Pfad in $G[V_X^Y \setminus (X \setminus S)]$ ersetzt werden muss. Bei der theoretischen Beschreibung wurde dazu der Pfad $uvXS$ gespeichert. Anhand $uvXS$ wurde dann mit dem Algorithmus Traceback-k ein kürzester (u, v)-Pfad in $G[V_X^Y \setminus (X \setminus S)]$ konstruiert. Im Gegensatz zu der theoretischen Beschreibung wurde bei der Implementierung nur die Länge $d(uvXS)$ und nicht der

Pfad $uvXS$ selbst gespeichert. Die künstliche Kante uv wurde daher durch einen kürzesten relevanten (u, v)-Pfad in $G[V_X^Y \setminus (X \setminus S)]$ wie folgt ersetzt: Es wurde der induzierte Teilgraph $G[V_X^Y \setminus (X \setminus S)]$ konstruiert und dann der kürzeste (u, v)-Pfad in diesem Teilgraphen mit Dijkstra berechnet. Diese Implementierung ist zwar einfach zu implementieren, aber theoretisch gesehen bezüglich der Laufzeit ineffizient. Das wichtigste ist, dass diese Implementierung die Güte des Algorithmus nicht ändert, denn wir haben sowohl bei der theoretischen Beschreibung als auch bei der Implementierung eine künstliche Kante uv mit Kosten c'_{uv} durch einen kürzesten (u, v)-Pfad mit Kosten c'_{uv} in $G[V_X^Y \setminus (X \setminus S)]$ ersetzt. Nur die Rechenwege sind unterschiedlich.

3. Nachdem die Heuristik alle künstlichen Kanten des berechneten Steinerbaums H durch die entsprechenden Pfade ersetzt hat, wird zum Schluss ein Spannbaum auf H berechnet. Bei der theoretischen Beschreibung haben wir dazu den DFS Algorithmus benutzt. Bei der Implementierung jedoch wurde der Spannbaum mit Hilfe der SAGE Methode `min_spanning_tree()` berechnet. Damit diese Methode einen Spannbaum und nicht einen MST berechnet, kann diese Methode wie folgt aufgerufen werden:

```
min_spanning_tree( weight_function = lambda e : 1 );
```

Die Option `weight_function = lambda e : 1` setzt die Kantenkosten $c_e := 1$ für jede Kante e von H. Das heißt, die Kantenkosten werden ignoriert. Somit wird ein Spannbaum und nicht unbedingt ein MST berechnet. Man beachte, dass wir die Güte der Heuristik nicht geändert haben. Denn wir haben sowohl bei der theoretischen Beschreibung als auch bei der Implementierung einen Spannbaum und keinen MST auf H berechnet. Nur die Rechenwege sind unterschiedlich.

4.1.3 Das Programm

Das Programm besteht aus drei Python Modulen:

1. `SteinLibParser.py` ist zum Parsen der Dateien in *SteinLib*-Format [1] zuständig.
2. `tree_decomposition.py` ist für die Baumzerlegung zuständig.
3. `stp_heuristic.py` enthält die Methode `stp_heuristic( SteinLib_File )`, wobei `SteinLib_File` eine Datei in SteinLib-Format ist. Diese Methode gibt ein Tupel (H, E_H) zurück, wobei H die Kosten und E_H die Kantenmenge des von der Heuristik berechneten Steinerbaums ist.

Möchte man nur die Kosten des von der Heuristik berechneten Steinerbaums beispielsweise in der Linux-Kommandozeile ausgeben, kann man das beiliegende Programm `main.py` wie folgt benutzen:

```
$ sage main.py <SteinLib_File>;
```

Dabei ist `<SteinLib_File>` eine Datei in SteinLib-Format.

Testset	Instanzen	r_{min}	r_{max}	μ_r	s^2
SP	5	1.0	1.0	1.0	0.0
WRP4	5	1.0	<1.0001	<1.0001	<0.0001
WRP3	13	1.0	<1.0001	<1.0001	<0.0001
GAP	2	1.0	1.0079	1.0039	<0.0001
TSPFST	44	1.0	1.098	1.0259	0.0005
MSM	5	1.0	1.0579	1.0304	0.0005
ALUT	1	1.0313	1.0313	1.0313	-
DIW	4	1.0	1.1291	1.0359	0.0039
ESxFST	145	1.0	1.109	1.0428	0.0007
P6E	15	1.0	1.1383	1.0445	0.002
I080	10	1.0214	1.0853	1.0493	0.0003
LIN	4	1.0	1.1803	1.054	0.0074
B	11	1.0	1.1721	1.0667	0.0032
P6Z	15	1.0	1.1896	1.0715	0.0033
DMXA	1	1.08	1.08	1.08	-
TAQ	1	1.119	1.119	1.119	-
Alle Testsets	281	1.0	1.1896	1.0394	0.0013

Tabelle 4.1: Bearbeitete Testsets, Anzahl der bearbeiteten Instanzen, minimale (bzw. maximale) beobachtete Güte r_{min} (bzw. r_{max}), durchschnittliche Güte μ_r, Stichprobenvarianz s^2 der Güte-Verteilung der Testsets

4.2 Tests

Die Heuristik wurde anhand der Steinerbaumproblem-Instanzen der SteinLib-Webseite [2] getestet. Wie wir in Abschnitt 4.1.2 bereits erwähnt haben, werden wir bei den Experimenten nur die Güte, aber nicht die Laufzeit der Heuristik untersuchen.

Im Folgenden bezeichnen wir mit OPT den Wert der optimalen Lösung und mit H den Wert, der von der Heuristik berechneten Lösung. Obige Webseite enthält auch die OPT-Werte der Instanzen. Das ermöglicht uns die experimentelle Güte $r := \frac{H}{OPT}$ der Heuristik festzustellen. Die Instanzen der SteinLib sind in sogenannten *Testsets* eingeteilt. Wegen der geringen Ausführungsgeschwindigkeit von Python-Programmen wurden Testsets mit mittelgroßen Instanzen untersucht. Dazu wurden zuerst alle Testsets außer folgenden ausgewählt:

- 1R, 2R, GENE (Parser fehlt);
- Relay (OPT-Werte fehlen);

Aus diesen ausgewählten Testsets wurden weiterhin die Instanzen mit $|E| \leq 4000$ Kanten und $tw \leq 15$ (Bemerkung 4.1.1) ausgewählt und untersucht. Mit dieser Eigenschaft

wurden 299 Instanzen gefunden. Von diesen Instanzen konnten 281 Instanzen bearbeitet werden. Die restlichen 18 Instanzen konnten wegen der nicht ausreichenden Kapazität des Hauptspeichers nicht bearbeitet werden. In Tabelle 4.1 sind die bearbeiteten Testsets und die Anzahl der bearbeiteten Instanzen in diesen Testsets aufgelistet. Dabei sind die Testsets ES10FST, ES20FST, ..., ES10000FST unter name ESxFST zusammengefasst. Die minimale (bzw. maximal) beobachtete Güte r_{min} (bzw. r_{max}), die durchschnittliche Güte μ_r und die Stichprobenvarianz s^2 [8] der Güte-Verteilung jedes Testsets ist auch in dieser Tabelle enthalten, wobei die Tabelle nach der durchschnittlichen Güte sortiert ist. Die maximal beobachtete Güte beträgt $r < 1.19$. Die durchschnittliche Güte aller Instanzen ist $\mu_r < 1.04$. Die Testergebnisse dieser Instanzen kann man aus dem Anhang A entnehmen.

Wir werden die folgende Zusammenhänge untersuchen:

- Ist die Güte der Heuristik von der Baumweite tw abhängig?
- Ist die Güte der Heuristik von der Größe des Graphen, genauer gesagt, von der Knotenanzahl abhängig?
- Wie ändert sich die Güte, wenn die Anzahl $|T|$ der Terminalknoten gegen die Knotenanzahl $|V|$ konvergiert?

Die großen Unterschiede (siehe Tabelle 4.1) zwischen den Testergebnissen der verschiedenen Testsets legen es nahe, die Testsets sowohl zusammen als auch im einzelnen zu untersuchen. Die meistens Testsets in Tabelle 4.1 haben zu wenig Instanzen. Daher werden wir hier nur die Testsets mit mindestens 10 Instanzen betrachten. Wir beschreiben unsere Beobachtungen für P6E, sie gelten analog für P6Z.

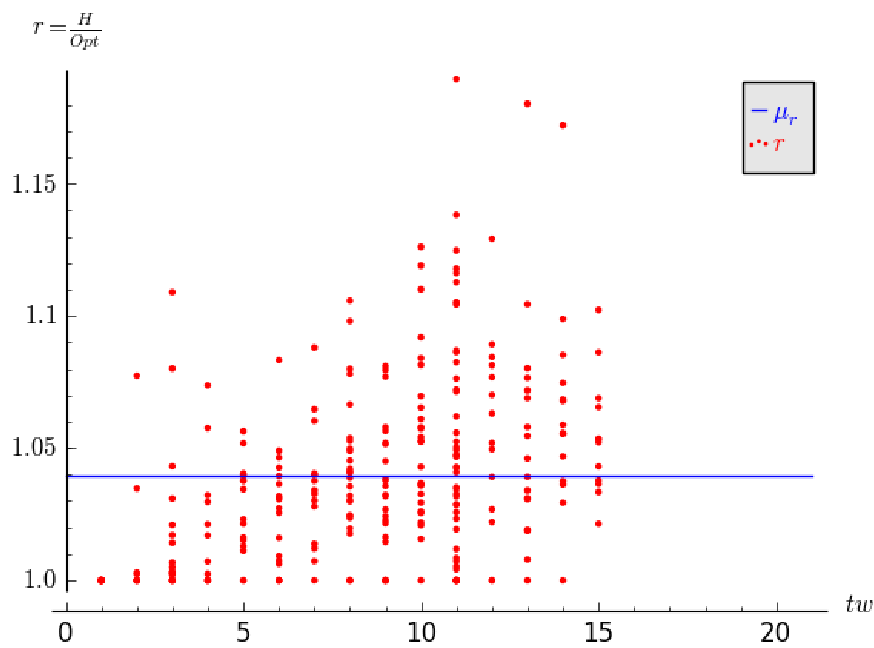
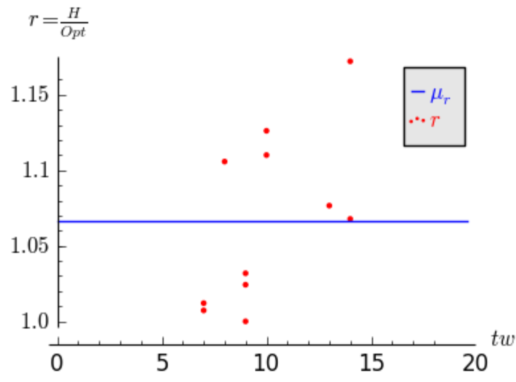


Abbildung 4.1: Baumweite tw und Güte r aller 281 Instanzen

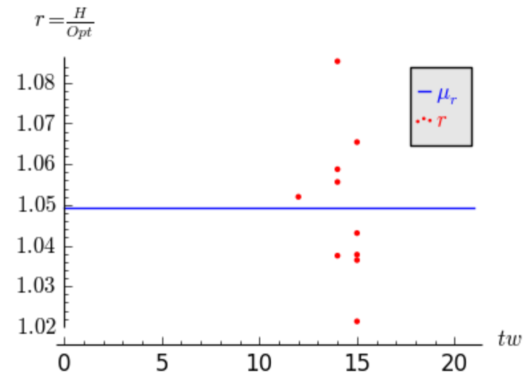
4.2.1 Zusammenhang: Baumweite $tw \leftrightarrow$ Güte r

Wir untersuchen den Zusammenhang zwischen der Güte und der Baumweite. In Abbildung 4.1 sind die Baumweite tw und die Güte r aller 281 Instanzen aus der Tabelle 4.1 als rote Punktwolke dargestellt, wobei der Punkt an Position (tw, r) eine Instanz mit Güte r und Baumweite tw (Bemerkung 4.1.1) vertritt. Die blaue Linie zeigt den Mittelwert $\mu_r = 1.04$ der Güte aller Instanzen. Die Güte r dieser Instanzen liegen im Intervall $[1, 1.19]$. Sowohl die Streuung der Güte, als auch die maximal beobachtete Güte scheint bis $tw = 11$ wachsend und danach fallend zu sein. Das kann auch daran liegen, dass die meistens Instanzen $tw = 11$ haben. Außerdem scheint die minimal beobachtete Güte unverändert zu bleiben.

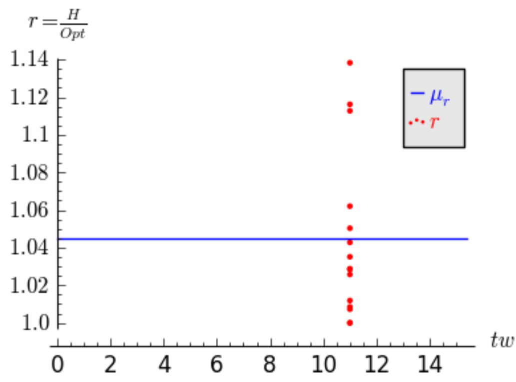
Wie wir oben erwähnt haben, werden wir die großen Testsets auch im einzelnen untersuchen. In Abbildung 4.2 sind die Punktwolken der Testsets B, I080, P6E, ESxFST, TSPFST und WRP3 zu sehen. Auch hier vertritt der rote Punkt an Position (tw, r) eine Instanz mit Güte r und Baumweite tw und die blaue Linie in Abbildung eines Testsets zeigt den Mittelwert der Güte aller Instanzen dieses Testsets. Beobachtet man die einzelnen Testsets (außer P6E, da die P6E-Instanzen nur die Baumweite $tw = 11$ haben), so scheint, dass bei wachsender Baumweite tw auch die Güte r wächst. Das kann man insbesondere in Abbildungen 4.2(d), 4.2(e) und 4.2(f) deutlich erkennen. Außerdem scheint die Güte bei den Testsets I080 (Abbildung 4.2(b)), TSPFST (Abbildung 4.2(e)) und WRP3 (Abbildung 4.2(f)) bei größeren tw stärker zu streuen als bei kleineren. Das scheint, bei den Testset ESxFST (Abbildung 4.2(d)) umgekehrt zu sein.



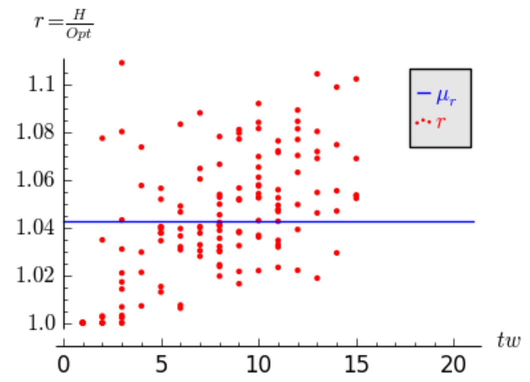
(a) Testset B



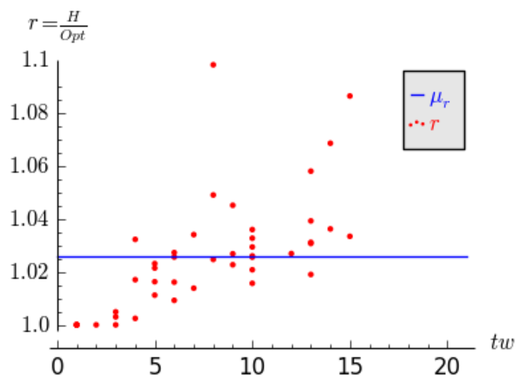
(b) Testset I080



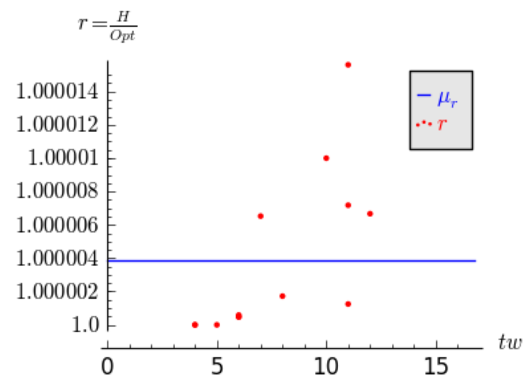
(c) Testset P6E



(d) Testset ESxFST



(e) Testset TSPFST



(f) Testset WRP3

Abbildung 4.2: Baumweite tw und Güte r der einzelnen Testsets

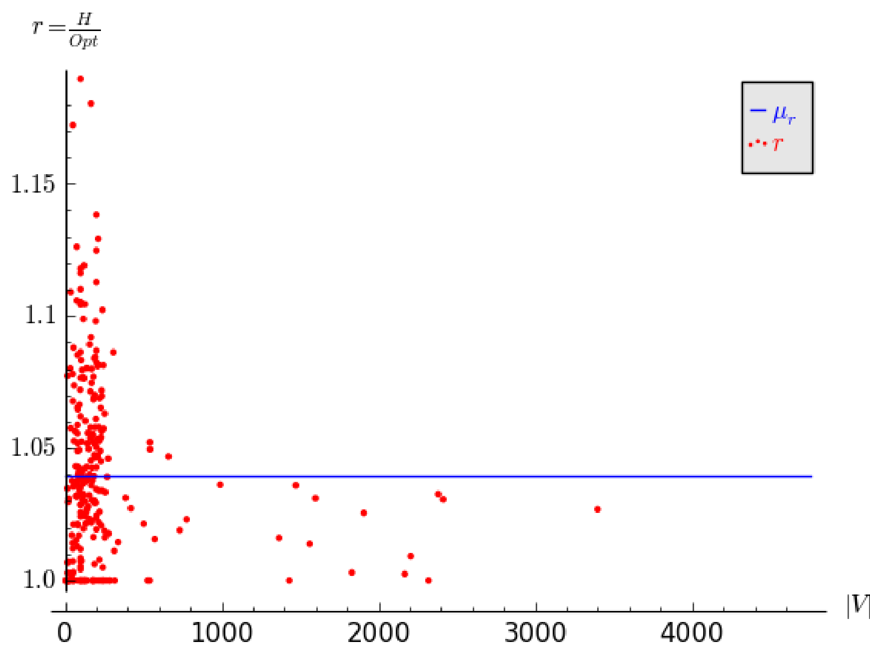


Abbildung 4.3: Knotenanzahl $|V|$ und Güte r aller 281 Instanzen

4.2.2 Zusammenhang: Knotenanzahl $|V| \leftrightarrow$ Güte r

Wir untersuchen den Zusammenhang zwischen der Güte und der Knotenanzahl. In Abbildung 4.3 sind die Knotenanzahl $|V|$ und die Güte r aller 281 Instanzen aus der Tabelle 4.1 dargestellt. Der rote Punkt an Position $(|V|, r)$ zeigt die Güte r einer Instanz mit Knotenanzahl $|V|$. Die blaue Linie zeigt den Mittelwert $\mu_r = 1.04$ der Güte aller Instanzen. In dieser Abbildung scheint die Güte bei kleinerer Knotenanzahl weiter zu streuen als bei größeren. Außerdem sieht es hier so aus, als würde die maximal beobachtete Güte mit wachsendem $|V|$ fallen, während die minimal beobachtete Güte unverändert bleibt. Diese Beobachtungen können natürlich auch daran liegen, dass die meistens Instanzen kleine Knotenanzahl haben.

Wir werden auch hier dieselben Testsets aus Abschnitt 4.2.1 betrachten. In Abbildung 4.4 sind die Punktwolken der Testsets B, I080, P6E, ESxFST, TSPFST und WRP3 zu sehen. Der rote Punkt an Position $(|V|, r)$ vertritt eine Instanz mit Güte r und Knotenanzahl $|V|$ und die blaue Linie in Abbildung eines Testsets zeigt den Mittelwert der Güte aller Instanzen dieser Testsets. In Testsets B (Abbildung 4.4(a)), ESxFST (Abbildung 4.4(d)) und TSPFST (Abbildung 4.4(e)) scheint die Güte bei wachsender Knotenanzahl weniger zu streuen und die maximal beobachtete Güte zu fallen. Außerdem scheint die minimal beobachtete Güte in den Testsets B (Abbildung 4.4(a)), P6E (Abbildung 4.4(c)), ESxFST (Abbildung 4.4(d)) und WRP3 (Abbildung 4.4(f)) bei steigender Knotenanzahl auch zu steigen, während sie in Testset TSPFST (Abbildung 4.4(e)) unverändert bleibt.

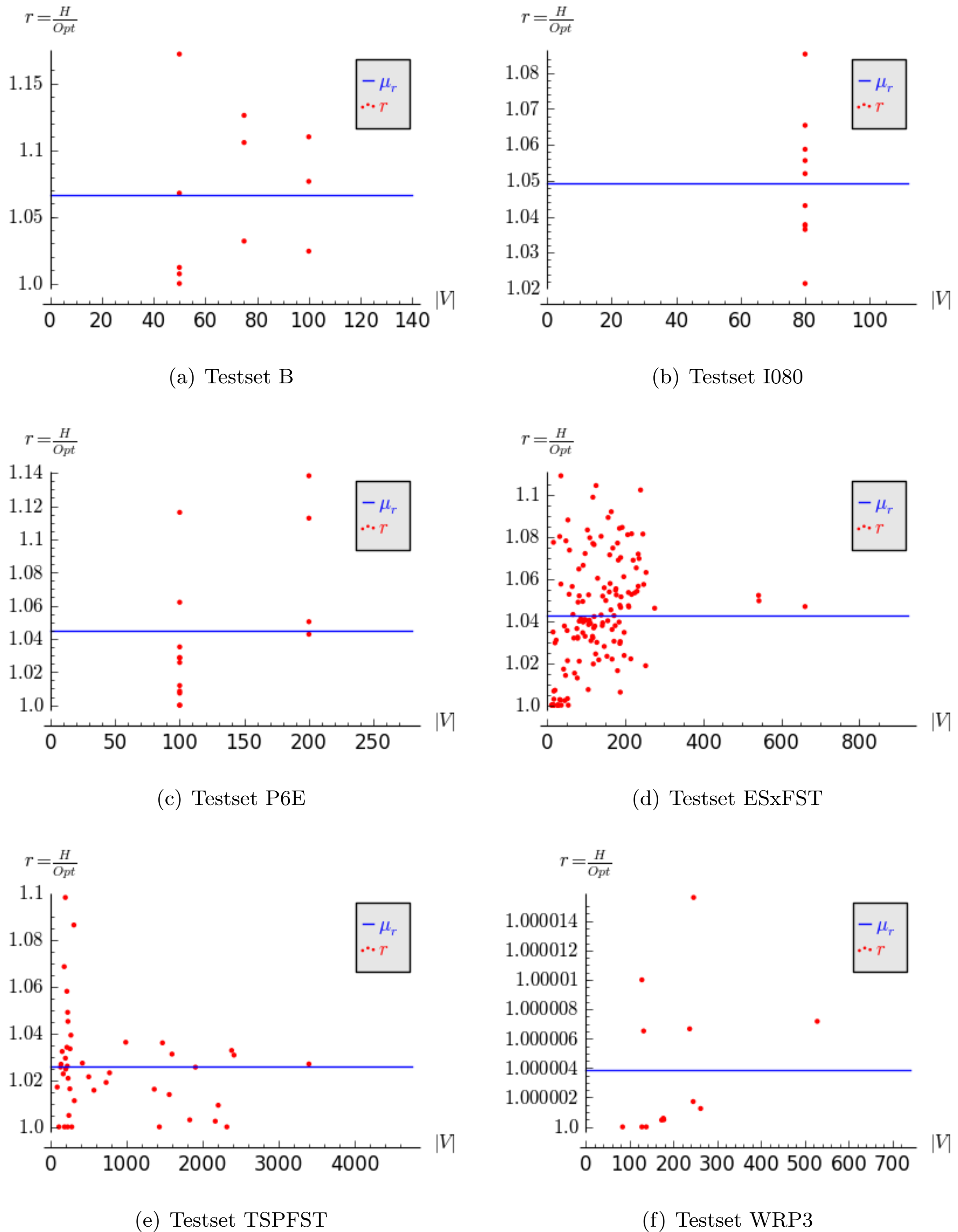


Abbildung 4.4: Knotenanzahl $|V|$ und Güte r der einzelnen Testsets

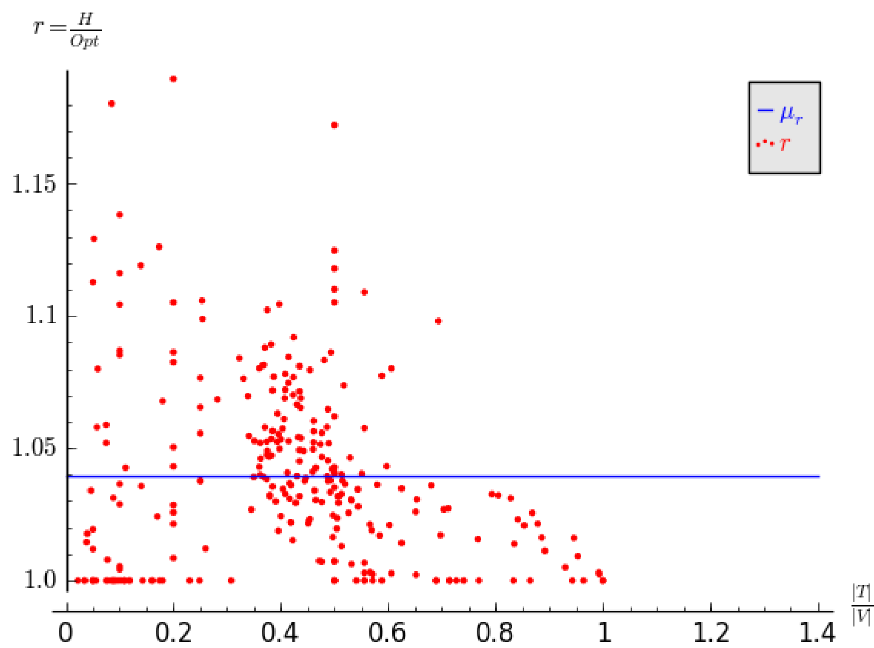
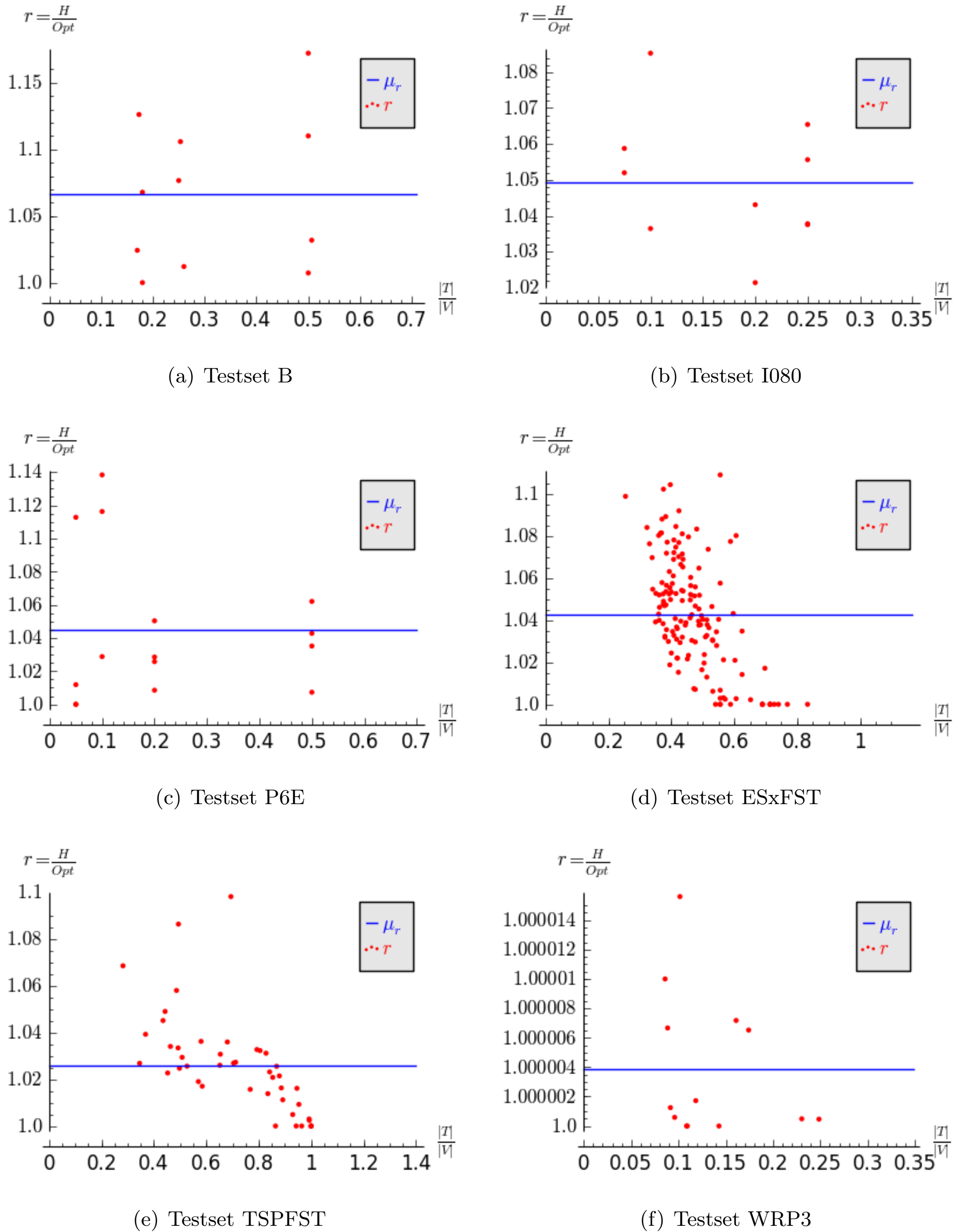


Abbildung 4.5: Terminal-Dichte $\frac{|T|}{|V|}$ und Güte r aller 281 Instanzen

4.2.3 Zusammenhang: $\frac{|T|}{|V|} \leftrightarrow$ Güte r

Wir bezeichnen den Wert $\frac{|T|}{|V|}$ als Terminal-Dichte und untersuchen den Zusammenhang zwischen diesem Wert und der Güte. In Abbildung 4.5 sind die Terminal-Dichte und die Güte r aller 281 Instanzen aus der Tabelle 4.1 dargestellt. Der rote Punkt an Position $(\frac{|T|}{|V|}, r)$ zeigt die Güte r einer Instanz mit Terminal-Dichte $\frac{|T|}{|V|}$. Die blaue Linie zeigt den Mittelwert $\mu_r = 1.04$ der Güte aller Instanzen. Die Beobachtung legt nahe, dass die Güte, aber auch die Streuung der Güte bei wachsender Terminal-Dichte fällt. Man könnte das wie folgt interpretieren: Je mehr Terminalknoten in einer Instanz enthalten sind, desto weniger Fehler kann die Heuristik bezüglich dieser Instanz machen. Denn alle Terminalknoten T müssen unbedingt gewählt werden. Der Fehler kann quasi nur bezüglich der restlichen Knoten aus der Menge $V \setminus T$ passieren und je größer die Terminal-Dichte ist, desto weniger Knoten bleiben in $V \setminus T$ übrig.

Wir werden auch hier dieselben Testsets aus Abschnitt 4.2.1 betrachten. In Abbildung 4.6 sind die Punktwolken der Testsets B, I080, P6E, ESxFST, TSPFST und WRP3 zu sehen. Der rote Punkt an Position $(\frac{|T|}{|V|}, r)$ zeigt die Güte r einer Instanz mit Terminal-Dichte $\frac{|T|}{|V|}$ und die blaue Linie in Abbildung eines Testsets zeigt den Mittelwert der Güte aller Instanzen dieses Testsets. In Testsets P6E (Abbildung 4.6(c)), ESxFST (Abbildung 4.6(d)), TSPFST (Abbildung 4.6(e)) und WRP3 (Abbildung 4.6(f)) scheint die Güte bei wachsender Terminal-Dichte zu fallen und weniger zu streuen. Besonders auffällig ist dies in Testsets ESxFST und TSPFST, da ihre Terminal-Dichte in Vergleich zu anderer Testsets aus einem etwas größeren Intervall stammt.



(a) Testset B

(b) Testset I080

(c) Testset P6E

(d) Testset ESxFST

(e) Testset TSPFST

(f) Testset WRP3

Abbildung 4.6: Terminal-Dichte $\frac{|T|}{|V|}$ und Güte r der einzelnen Testsets

4.2.4 Fazit

Wir haben bei den Experimenten Folgendes beobachtet:

- Die Güte der Heuristik scheint sich mit wachsender Knotenanzahl zu bessern. Das kann natürlich auch daran liegen, dass die meistens untersuchten Instanzen kleine Knotenanzahl haben.
- Die Experimente legen nahe, dass je näher die Anzahl $|T|$ der Terminalknoten an $|V|$ kommt, desto besser die Güte der Heuristik wird. Die Heuristik scheint also für Instanzen geeignet zu sein, bei denen die meisten Knoten Terminalknoten sind.
- Außerdem scheint die Heuristik nicht für Instanzen mit zu großer Baumweite geeignet zu sein, da die Güte schlechter wird. Hier eignet sich der Algorithmus allerdings ohnehin nicht gut, da die Laufzeit exponentiell in der Baumweite ist.

Kapitel 5

Zusammenfassung und Ausblick

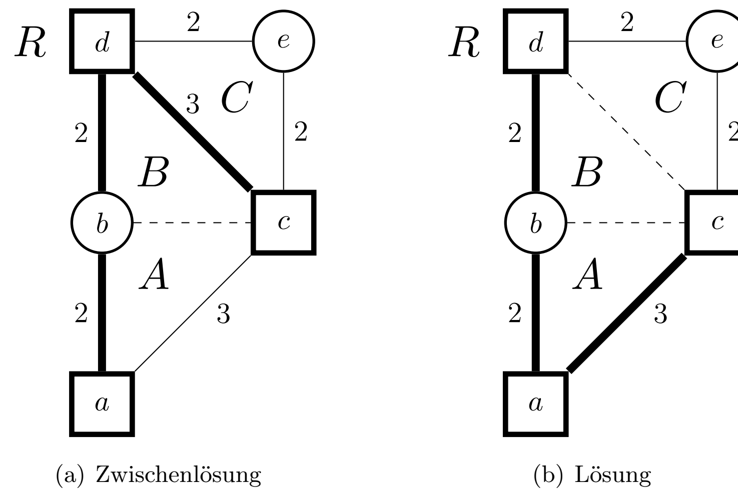
In dieser Arbeit haben wir für das Steinerbaumproblem eine baumzerlegungsbasierte Heuristik mit der Laufzeit $F + O\left(|V| \cdot 2^{tw} \cdot tw^3\right)$ vorgestellt, wobei V die Knotenmenge des Graphen G, tw die Baumweite einer Baumzerlegung von G und F eine Zufallsvariable mit Erwartungswert $\mathbf{E}(F) = O\left(|V| \cdot 2^{tw} \cdot tw\right)$ ist. Dabei ist die Wahrscheinlichkeit, dass F größer als $|V| \cdot 2^{tw} \cdot tw^3$ ist, ist kleiner als $2^{-|V| \cdot tw^2}$.

Die dominante Faktor 2^{tw} ist üblich für baumzerlegungsbasierte Algorithmen und entsteht wegen der Kombinationen der Teillösungen. Meistens kostet diese Kombination $O((2^{tw})^2)$ Zeit, da man zwei Tabellen jeweils mit $O(2^{tw})$ Zeilen kombiniert, und dabei die Kombinationsmöglichkeit jeder Zeile einer Tabelle mit jeder Zeile der zweiten Tabelle betrachtet. Dank schneller Verkettung kann unser Algorithmus die Kombination zweier Tabellen in $O(2^{tw})$ Zeit durchführen.

Die experimentelle Güte der Heuristik wurde anhand der Steinerbaumproblem-Instanzen aus der Webseite [2] untersucht. Die größte beobachtete Güte bei diesen Tests war 1.19 und die durchschnittliche Güte war 1.04. Außerdem legen die Experimente (Abschnitt 4.2.4) nahe, dass die Heuristik insbesondere für den Instanzen gut geeignet ist, bei denen die meisten Knoten Terminalknoten sind.

5.1 Ein weiterer Lösungsansatz

Wir können den Trick aus Abschnitt 3.4.1 noch verbessern. Siehe die Abbildung 3.13. Damit **H*** gilt, versucht Phase 1 die Komponente mit Knoten d, nämlich $(\{d, b, a\}, \{db, ba\})$ und die Komponente mit Knoten c, nämlich $(\{c\}, \emptyset)$ zusammenzuhängen. Dieser Zusammenhang muss nicht unbedingt durch einen (c, d)-Pfad hergestellt werden. Das könnte genauso gut ein (c, b)-Pfad, oder (c, a)-Pfad sein. Im Beispiel wäre der (c, a)-Pfad mit der Länge 3 die beste Wahl. Wir erzeugen eine künstliche Kante cd mit Kosten 3 und verbinden die beide Knoten(Abbildung 5.1(a)). Das führt zu einer Lösung mit Kosten $7(= 2 + 2 + 3)$ (Abbildung 5.1(b)).



(a) Zwischenlösung (b) Lösung

Abbildung 5.1: Verbesserter Trick mit künstlicher Kante cd

Das ist auch die Kernidee von Prim. Prim verwaltet einen sogenannten *zu MST erweiterbaren Baum* B und fügt zu B jedes Mal einen neuen Knoten hinzu, der zu B am nächsten liegt. Der Punkt ist, dass der neue Knoten nicht zu einem bestimmten sondern irgendeinen B Knoten am nächsten liegen muss. Es hat sich herausgestellt, dass der durch diesen Trick konstruierte Steinerbaum kreisfrei ist und keine Kante mehrfach enthält, während das für den Trick aus dem Abschnitt 3.4.1 nicht gilt. Aus Zeitgründen wurde jedoch dieser Trick nicht verfolgt.

Anhang A

Testergebnisse

Die Testergebnisse der einzelnen Testsets aus Tabelle 4.1:

Testset SP

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	antiwheel5	10	15	5	4	0.50	7	7	1.0
2	design432	8	20	4	5	0.50	9	9	1.0
3	oddcycle3	6	9	3	2	0.50	4	4	1.0
4	oddwheel3	7	9	4	3	0.57	5	5	1.0
5	se03	13	21	4	6	0.31	12	12	1.0

Testset WRP4

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	wrp4-11	123	233	11	12	0.09	1100188	1100179	1.00000818049
2	wrp4-13	110	188	13	9	0.12	1300805	1300798	1.00000538131
3	wrp4-15	193	369	15	14	0.08	1500405	1500405	1.0
4	wrp4-18	211	380	18	13	0.09	1801471	1801464	1.00000388573
5	wrp4-19	119	206	19	4	0.16	1901446	1901446	1.0

Testset WRP3

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	wrp3-11	128	227	11	10	0.09	1100372	1100361	1.00000999672
2	wrp3-12	84	149	12	4	0.14	1200237	1200237	1.0
3	wrp3-14	128	247	14	5	0.11	1400250	1400250	1.0
4	wrp3-15	138	257	15	4	0.11	1500422	1500422	1.0

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
5	wrp3-17	177	354	17	6	0.10	1700443	1700442	1.00000058808
6	wrp3-21	237	444	21	12	0.09	2100536	2100522	1.00000666501
7	wrp3-23	132	230	23	7	0.17	2300260	2300245	1.00000652104
8	wrp3-24	262	487	24	11	0.09	2400626	2400623	1.00000124968
9	wrp3-25	246	468	25	11	0.10	2500579	2500540	1.00001559663
10	wrp3-29	245	436	29	8	0.12	2900484	2900479	1.00000172385
11	wrp3-41	178	307	41	6	0.23	4100468	4100466	1.00000048775
12	wrp3-43	173	298	43	6	0.25	4300459	4300457	1.00000046507
13	wrp3-85	528	1017	85	11	0.16	8500800	8500739	1.00000717585

Testset GAP

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	gap1500	220	374	17	13	0.08	256	254	1.00787401575
2	gap2975	179	293	10	9	0.06	245	245	1.0

Testset TSPFST

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	a280fst	314	328	280	5	0.89	2530	2502	1.01119104716
2	att48fst	139	202	48	9	0.35	31046	30236	1.02678925784
3	berlin52fst	89	104	52	4	0.58	6875	6760	1.01701183432
4	bier127fst	258	357	127	15	0.49	107764	104284	1.03337041157
5	d1291fst	1365	1456	1291	6	0.95	489171	481421	1.01609817602
6	d1655fst	1906	2083	1655	10	0.87	599873	584948	1.02551508852
7	d198fst	232	256	198	10	0.85	131858	129175	1.02077027289
8	d2103fst	2206	2272	2103	6	0.95	776909	769797	1.00923879932
9	eil51fst	181	289	51	14	0.28	437	409	1.0684596577
10	fl1577fst	2413	3412	1577	13	0.65	20433723	19825626	1.03067227234
11	fl417fst	732	1084	417	13	0.57	11089863	10883190	1.01899011227
12	kroA100fst	197	250	100	10	0.51	21000	20401	1.02936130582
13	kroB100fst	230	313	100	9	0.43	22167	21211	1.04507095375
14	kroD100fst	216	288	100	7	0.46	21132	20437	1.03400694818
15	kroE100fst	226	306	100	8	0.44	22285	21245	1.04895269475
16	lin105fst	216	323	105	13	0.49	14207	13429	1.05793432125
17	p654fst	777	867	654	5	0.84	322200	314925	1.02310073827
18	pcb442fst	503	531	442	5	0.88	48696	47675	1.02141583639

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
19	pr1002fst	1473	1715	1002	10	0.68	251909	243176	1.03591226108
20	pr107fst	111	110	107	1	0.96	34850	34850	1.0
21	pr124fst	154	165	124	4	0.81	54459	52759	1.03222199056
22	pr136fst	196	250	136	8	0.69	95321	86811	1.09802905162
23	pr144fst	221	285	144	10	0.65	54300	52925	1.0259801606
24	pr152fst	308	431	152	15	0.49	69872	64323	1.08626774249
25	pr226fst	255	269	226	5	0.89	71850	70700	1.01626591231
26	pr2392fst	3398	3966	2392	12	0.70	368635	358989	1.02686990409
27	pr264fst	280	287	264	3	0.94	41400	41400	1.0
28	pr299fst	420	500	299	6	0.71	45888	44671	1.02724362562
29	pr439fst	572	662	439	10	0.77	98925	97400	1.01565708419
30	pr76fst	168	247	76	9	0.45	98081	95908	1.02265712975
31	rat99fst	269	399	99	13	0.37	1273	1225	1.03918367347
32	rd100fst	201	253	100	8	0.50	783119524	764269099	1.02466464368
33	rl1304fst	1562	1694	1304	7	0.83	239921	236649	1.01382638422
34	rl1323fst	1598	1750	1323	13	0.83	261509	253620	1.03110559104
35	rl1889fst	2382	2674	1889	10	0.79	304850	295208	1.03266171648
36	st70fst	133	169	70	6	0.53	642	626	1.02555910543
37	ts225fst	225	224	225	1	1.00	1120	1120	1.0
38	tsp225fst	242	252	225	3	0.93	358600	356850	1.0049040213
39	u1432fst	1432	1431	1432	1	1.00	1465	1465	1.0
40	u159fst	184	186	159	2	0.86	390	390	1.0
41	u1817fst	1831	1846	1817	3	0.99	5529561	5513053	1.00299434814
42	u2152fst	2167	2184	2152	4	0.99	6268543	6253305	1.00243679142
43	u2319fst	2319	2318	2319	1	1.00	2322	2322	1.0
44	u574fst	990	1258	574	14	0.58	3636179	3509275	1.03616245521

Testset MSM

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	msm1707	278	478	11	8	0.04	574	564	1.01773049645
2	msm1844	90	135	10	6	0.11	196	188	1.04255319149
3	msm4038	237	390	11	13	0.05	365	353	1.03399433428
4	msm4224	191	302	11	9	0.06	329	311	1.0578778135
5	msm4414	317	476	11	8	0.03	408	408	1.0

Testset ALUT

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	alut2764	387	626	34	6	0.09	660	640	1.03125

Testset DIW

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	diw0260	539	985	12	8	0.02	468	468	1.0
2	diw0393	212	381	11	12	0.05	341	302	1.12913907285
3	diw0460	339	579	13	9	0.04	350	345	1.01449275362
4	diw0540	286	465	10	11	0.03	374	374	1.0

Testset ESxFST

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	es10fst01	18	20	10	3	0.56	22920745	22920745	1.0
2	es10fst02	14	13	10	1	0.71	19134104	19134104	1.0
3	es10fst03	17	20	10	2	0.59	26003678	26003678	1.0
4	es10fst04	18	20	10	2	0.56	20519872	20461116	1.00287159312
5	es10fst05	12	11	10	1	0.83	18818916	18818916	1.0
6	es10fst06	17	20	10	2	0.59	28594811	26540768	1.07739199559
7	es10fst07	14	13	10	1	0.71	26025072	26025072	1.0
8	es10fst08	21	28	10	4	0.48	25798844	25056214	1.02963855593
9	es10fst09	21	29	10	4	0.48	22219167	22062355	1.00710767278
10	es10fst10	18	21	10	3	0.56	24096332	23936095	1.00669436681
11	es10fst11	14	13	10	1	0.71	22239535	22239535	1.0
12	es10fst12	13	12	10	1	0.77	19626318	19626318	1.0
13	es10fst13	18	21	10	3	0.56	19483914	19483914	1.0
14	es10fst14	24	32	10	3	0.42	22532068	21856128	1.03092679545
15	es10fst15	16	18	10	2	0.62	19290225	18641924	1.03477650697
16	es20fst01	29	28	20	1	0.69	33703886	33703886	1.0
17	es20fst02	29	28	20	1	0.69	32639486	32639486	1.0
18	es20fst03	27	26	20	1	0.74	27847417	27847417	1.0
19	es20fst04	57	83	20	8	0.35	29080365	27624394	1.05270598877
20	es20fst05	54	77	20	7	0.37	37028588	34033163	1.08801488713
21	es20fst06	29	28	20	1	0.69	36014241	36014241	1.0
22	es20fst07	45	59	20	5	0.44	36249542	34934874	1.03763196627

N	Instanz	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	tw	$\lvert T\rvert/\lvert V\rvert$	H	OPT	Güte
23	es20fst08	52	74	20	8	0.38	39364551	38016346	1.0354638239
24	es20fst09	36	42	20	3	0.56	40745333	36739939	1.10902015923
25	es20fst10	49	67	20	8	0.41	36681930	34024740	1.07809582086
26	es20fst11	33	36	20	3	0.61	27196633	27123908	1.00268121393
27	es20fst12	33	36	20	3	0.61	32892337	30451397	1.08015855562
28	es20fst13	35	40	20	2	0.57	34522183	34438673	1.00242489018
29	es20fst14	36	44	20	4	0.56	36023119	34062374	1.05756336889
30	es20fst15	37	43	20	2	0.54	32303746	32303746	1.0
31	es30fst01	79	115	30	11	0.38	41986714	40692993	1.03179223018
32	es30fst02	71	97	30	5	0.42	41522510	40900061	1.01521877926
33	es30fst03	83	120	30	7	0.36	44841925	43120444	1.03992261768
34	es30fst04	80	115	30	6	0.38	44214789	42150958	1.0489628492
35	es30fst05	58	71	30	4	0.52	44815270	41739748	1.07368329104
36	es30fst06	83	119	30	9	0.36	42031799	39955139	1.05197479103
37	es30fst07	53	64	30	4	0.57	44687091	43761391	1.02115334954
38	es30fst08	69	93	30	6	0.43	43020761	41691217	1.031890266
39	es30fst09	43	44	30	3	0.70	37770155	37133658	1.01714070292
40	es30fst10	48	52	30	3	0.62	43291202	42686610	1.01416350467
41	es30fst11	79	112	30	9	0.38	42995641	41647993	1.03235805384
42	es30fst12	46	48	30	3	0.65	38500687	38416720	1.00218568894
43	es30fst13	65	84	30	5	0.46	39516544	37406646	1.05640436194
44	es30fst14	53	58	30	3	0.57	43030823	42897025	1.0031190508
45	es30fst15	118	188	30	14	0.25	47288801	43035576	1.09883044205
46	es40fst01	93	127	40	8	0.43	47821753	44841522	1.06646141494
47	es40fst02	82	105	40	7	0.49	49839909	46811310	1.06469801849
48	es40fst03	87	116	40	8	0.46	52027875	49974157	1.04109560067
49	es40fst04	55	55	40	2	0.73	45289864	45289864	1.0
50	es40fst05	121	180	40	11	0.33	55902516	51940413	1.07628169995
51	es40fst06	92	123	40	11	0.43	52206254	49753385	1.04930054508
52	es40fst07	77	95	40	6	0.52	47300737	45639009	1.03641025597
53	es40fst08	98	137	40	11	0.41	52259334	48745996	1.0720743915
54	es40fst09	107	153	40	9	0.37	53745512	51761789	1.03832408111
55	es40fst10	107	152	40	10	0.37	60133015	57136852	1.05243836325
56	es40fst11	97	135	40	8	0.41	48638565	46734214	1.04074854025
57	es40fst12	67	75	40	3	0.60	45734224	43843378	1.04312728823
58	es40fst13	78	95	40	5	0.51	52554879	51884545	1.01291972398
59	es40fst14	98	134	40	11	0.41	50777492	49166952	1.03275655566

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
60	es40fst15	93	129	40	5	0.43	52836612	50828067	1.03951645456
61	es50fst01	118	160	50	12	0.42	59172077	54948660	1.07686114639
62	es50fst02	125	177	50	8	0.40	56835834	55484245	1.02435987009
63	es50fst03	128	182	50	8	0.39	56324598	54691035	1.02986893556
64	es50fst04	106	138	50	6	0.47	51918307	51535766	1.00742282554
65	es50fst05	104	135	50	6	0.48	59782023	55186015	1.08328211414
66	es50fst06	126	182	50	13	0.40	61628529	55804287	1.10436907831
67	es50fst07	143	211	50	12	0.35	51916843	49961178	1.03914369273
68	es50fst08	83	96	50	3	0.60	54878357	53754708	1.02090326674
69	es50fst09	139	202	50	13	0.36	57746191	53456773	1.08024087051
70	es50fst10	139	207	50	10	0.36	56355658	54037963	1.04289012523
71	es50fst11	100	131	50	5	0.50	54653080	52532923	1.04035863377
72	es50fst12	110	149	50	9	0.45	57658910	53409291	1.07956703638
73	es50fst13	92	116	50	5	0.54	55747412	53891019	1.03444716827
74	es50fst14	120	167	50	10	0.42	55518241	53551419	1.03672772891
75	es50fst15	112	147	50	8	0.45	54205639	52180862	1.03880305772
76	es60fst01	123	159	60	7	0.49	55783103	53761423	1.03760465939
77	es60fst02	186	280	60	10	0.32	60020127	55367804	1.08402578148
78	es60fst03	113	142	60	6	0.53	58300533	56566797	1.03064935778
79	es60fst04	162	238	60	10	0.37	58573375	55371042	1.05783407507
80	es60fst05	119	148	60	8	0.50	55776948	54704991	1.01959523218
81	es60fst06	130	174	60	7	0.46	64065514	60421961	1.06030179987
82	es60fst08	109	133	60	7	0.55	60484830	58138178	1.04036335642
83	es60fst09	151	216	60	8	0.40	58656053	55877112	1.04973308212
84	es60fst10	133	177	60	9	0.45	58866454	57624488	1.02155274681
85	es60fst11	121	154	60	8	0.50	58505320	56141666	1.04210160062
86	es60fst12	176	257	60	13	0.34	63059347	59791362	1.05465647362
87	es60fst13	157	226	60	12	0.38	66672877	61213533	1.08918524601
88	es60fst14	118	149	60	8	0.51	57830229	56035528	1.03202791272
89	es60fst15	117	151	60	7	0.51	58475811	56622581	1.03272952181
90	es70fst01	154	209	70	11	0.45	63502124	62058863	1.0232563236
91	es70fst02	147	197	70	11	0.48	64327575	60928488	1.0557881397
92	es70fst03	181	264	70	9	0.39	66705873	61934664	1.07703616508
93	es70fst04	167	231	70	10	0.42	65203417	62938583	1.03598482667
94	es70fst05	169	231	70	14	0.41	66905747	62256993	1.07467039084
95	es70fst06	187	268	70	11	0.37	65074156	62124528	1.04747928226
96	es70fst07	167	230	70	10	0.42	63585661	62223666	1.02188869746

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
97	es70fst09	161	220	70	11	0.43	67485989	62986133	1.07144201089
98	es70fst10	165	225	70	10	0.42	68259735	62511830	1.09194907588
99	es70fst11	177	254	70	14	0.40	70135262	66455760	1.05536769123
100	es70fst12	142	181	70	9	0.49	65432144	63047132	1.03782903241
101	es70fst13	160	219	70	8	0.44	66298839	62912258	1.05383022495
102	es70fst14	143	184	70	5	0.49	63541963	60411124	1.05182553796
103	es70fst15	178	251	70	11	0.39	65584943	62318458	1.05241601132
104	es80fst01	187	255	80	14	0.43	73008099	70927442	1.02933500689
105	es80fst02	183	249	80	13	0.44	69773708	65273810	1.06893879797
106	es80fst03	189	261	80	12	0.42	69916288	65332546	1.07016016183
107	es80fst04	198	280	80	11	0.40	66413734	64193446	1.03458745617
108	es80fst05	172	228	80	8	0.47	68366417	66350529	1.03038239529
109	es80fst06	172	224	80	11	0.47	74030436	71007444	1.04257288856
110	es80fst07	193	271	80	12	0.41	73993652	68228475	1.08449810728
111	es80fst08	217	306	80	10	0.37	72946349	67452377	1.08144964261
112	es80fst09	236	343	80	10	0.34	74692425	69825651	1.06969894201
113	es80fst10	156	197	80	5	0.51	68125809	65497988	1.04012063699
114	es80fst11	209	295	80	11	0.38	69416774	66283099	1.0472771347
115	es80fst12	147	180	80	7	0.54	66886753	65070089	1.02791857254
116	es80fst13	164	211	80	8	0.49	71104939	68022647	1.04531273239
117	es80fst14	209	297	80	15	0.38	73831167	70077902	1.05355846698
118	es80fst15	197	282	80	10	0.41	74208740	69939071	1.06104840884
119	es90fst01	181	231	90	9	0.50	69471831	68350357	1.01640772703
120	es90fst02	221	313	90	15	0.41	76206144	71294845	1.06888715446
121	es90fst04	217	299	90	10	0.41	74642095	70910063	1.052630499
122	es90fst05	190	254	90	9	0.47	75529413	71831224	1.05148442132
123	es90fst06	215	290	90	12	0.42	70152858	68640346	1.02203532016
124	es90fst07	175	221	90	5	0.51	74758901	72036885	1.03778642011
125	es90fst08	234	332	90	9	0.38	76431987	72341668	1.05654167388
126	es90fst09	234	331	90	13	0.38	72729153	67856007	1.07181598528
127	es90fst10	246	356	90	12	0.37	78191587	72310409	1.08133238466
128	es90fst11	225	323	90	15	0.40	76169471	72310039	1.05337339121
129	es90fst12	207	284	90	9	0.43	74987193	69367257	1.08101712887
130	es90fst13	240	349	90	15	0.38	80255821	72810663	1.10225367677
131	es90fst14	185	243	90	6	0.49	71921996	69188992	1.03950056101
132	es100fst03	189	233	100	6	0.53	76125175	72746006	1.04645160863
133	es100fst04	188	235	100	7	0.53	76591482	74342392	1.03025312933

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
134	es100fst05	188	238	100	6	0.53	76138034	75670198	1.00618256609
135	es100fst07	276	401	100	13	0.36	81323296	77740576	1.04608558599
136	es100fst08	210	276	100	11	0.48	76444159	73033178	1.04670454023
137	es100fst09	248	342	100	10	0.40	82425401	77952027	1.05738624346
138	es100fst10	229	312	100	10	0.44	80908222	75952202	1.06525182772
139	es100fst11	253	362	100	13	0.40	80150632	78674859	1.01875787283
140	es100fst13	254	361	100	12	0.39	79311194	74604990	1.06308162497
141	es100fst14	198	253	100	8	0.51	80489868	78632795	1.023617029
142	es100fst15	231	319	100	10	0.43	74261653	70446493	1.05415684781
143	es250fst02	542	719	250	15	0.46	121161333	115150079	1.05220364634
144	es250fst03	543	727	250	12	0.46	120336188	114650399	1.04959240482
145	es250fst11	661	952	250	14	0.38	118181966	112889613	1.04688077901

Testset P6E

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	p619	100	180	5	11	0.05	7485	7485	1.0
2	p620	100	180	5	11	0.05	8746	8746	1.0
3	p621	100	180	5	11	0.05	8791	8688	1.01185543278
4	p622	100	180	10	11	0.10	17828	15972	1.11620335587
5	p623	100	180	10	11	0.10	20057	19496	1.02877513336
6	p624	100	180	20	11	0.20	20823	20246	1.02849945668
7	p625	100	180	20	11	0.20	23274	23078	1.008492937
8	p626	100	180	20	11	0.20	22921	22346	1.02573167457
9	p627	100	180	50	11	0.50	42075	40647	1.03513174404
10	p628	100	180	50	11	0.50	42490	40008	1.06203759248
11	p629	100	180	50	11	0.50	43602	43287	1.00727701157
12	p630	200	370	10	11	0.05	29070	26125	1.11272727273
13	p631	200	370	20	11	0.10	44469	39067	1.13827527069
14	p632	200	370	40	11	0.20	59046	56217	1.05032285608
15	p633	200	370	100	11	0.50	89957	86268	1.04276209023

Testset I080

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	i080-001	80	120	6	14	0.07	1892	1787	1.05875769446
2	i080-004	80	120	6	12	0.07	1963	1866	1.05198285102

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
3	i080-101	80	120	8	15	0.10	2703	2608	1.03642638037
4	i080-103	80	120	8	14	0.10	2825	2603	1.08528620822
5	i080-201	80	120	16	15	0.20	4965	4760	1.04306722689
6	i080-204	80	120	16	15	0.20	4588	4492	1.0213713268
7	i080-301	80	120	20	14	0.25	5826	5519	1.05562601921
8	i080-302	80	120	20	14	0.25	6167	5944	1.03751682369
9	i080-303	80	120	20	15	0.25	6155	5777	1.06543188506
10	i080-304	80	120	20	15	0.25	5797	5586	1.03777300394

Testset LIN

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	lin01	53	80	4	9	0.08	503	503	1.0
2	lin02	55	82	6	9	0.11	557	557	1.0
3	lin03	57	84	8	9	0.14	959	926	1.03563714903
4	lin06	165	274	14	13	0.08	1591	1348	1.18026706231

Testset B

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	b01	50	63	9	9	0.18	82	82	1.0
2	b02	50	63	13	7	0.26	84	83	1.01204819277
3	b03	50	63	25	7	0.50	139	138	1.00724637681
4	b04	50	100	9	14	0.18	63	59	1.06779661017
5	b06	50	100	25	14	0.50	143	122	1.17213114754
6	b07	75	94	13	10	0.17	125	111	1.12612612613
7	b08	75	94	19	8	0.25	115	104	1.10576923077
8	b09	75	94	38	9	0.51	227	220	1.03181818182
9	b13	100	125	17	9	0.17	169	165	1.02424242424
10	b14	100	125	25	13	0.25	253	235	1.07659574468
11	b15	100	125	50	10	0.50	353	318	1.11006289308

Testset P6Z

N	Instanz	$\|V\|$	$\|E\|$	$\|T\|$	tw	$\|T\|/\|V\|$	H	OPT	Güte
1	p602	100	180	5	11	0.05	8083	8083	1.0
2	p603	100	180	5	11	0.05	5022	5022	1.0

N	Instanz	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	tw	$\lvert T\rvert/\lvert V\rvert$	H	OPT	Güte
3	p604	100	180	10	11	0.10	12585	11397	1.10423795736
4	p605	100	180	10	11	0.10	10410	10355	1.00531144375
5	p606	100	180	10	11	0.10	13103	13048	1.0042152054
6	p607	100	180	20	11	0.20	16684	15358	1.08633936711
7	p608	100	180	20	11	0.20	17176	14439	1.18955606344
8	p609	100	180	20	11	0.20	20182	18263	1.10507583639
9	p610	100	180	50	11	0.50	33331	30161	1.10510261596
10	p611	100	180	50	11	0.50	30075	26903	1.11790506635
11	p612	100	180	50	11	0.50	31496	30258	1.04091479939
12	p613	200	370	10	11	0.05	18785	18429	1.01931738022
13	p614	200	370	20	11	0.10	29646	27276	1.08688957325
14	p615	200	370	40	11	0.20	45978	42474	1.0824975279
15	p616	200	370	100	11	0.50	70030	62263	1.12474503317

Testset DMXA

N	Instanz	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	tw	$\lvert T\rvert/\lvert V\rvert$	H	OPT	Güte
1	dmxa0628	169	280	10	8	0.06	297	275	1.08

Testset TAQ

N	Instanz	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	tw	$\lvert T\rvert/\lvert V\rvert$	H	OPT	Güte
1	taq0920	122	194	17	10	0.14	235	210	1.11904761905

Abbildungsverzeichnis

1.1 Netz und Gruppenbaum. 1

2.1 Ein Graph 3
2.2 Ein Steinerbaum (fette Kanten) 5
2.3 Ein Graph (links) und seine Baumzerlegung (rechts) 6
2.4 Teilbäume J_C^D, J_D^C und Teilgraphen G_C^D, G_D^C 7

3.1 Separator $C \cap D$ und Teilgraph H_C^D 9
3.2 Baumzerlegung mit künstlicher Kante DR 10
3.3 Die Teillösungen $tab_D(\{e, f\})$, $tab_C(\{c, e\})$, $tab_A(\{a, c\})$ und $tab_B(\{b, c, e\})$ 15
3.4 Konstruktion des Teilgraphen H_X^Y mit $V(H_X^Y) \cap X = X$ 18
3.5 Eine Steinerbaum-Instanz und seine Baumzerlegung 20
3.6 Phase 1 von A: Die Teilgraphen $G[S]$ und H_A^C (fette Kanten) mit $V(H_A^C) \cap A = S$ 20
3.7 Der Teillgraph $tab_C(P)$ nach der Phase 2 von A 21
3.8 Phase 1 von B: Die Teilgraphen $G[S]$ und H_B^C (fette Kanten) mit $V(H_B^C) \cap B = S$ 22
3.9 Der Teillgraph $tab_C(P)$ nach der Phase 2 von B 22
3.10 Phase 1 von C: Die Teilgraphen $G[S]$ und H_C^D (fette Kanten) mit $V(H_C^D) \cap C = S$ 23
3.11 Der Teillgraph $tab_D(P)$ nach der Phase 2 von C 24
3.12 Phase 1 von D: Die Teilgraphen $G[S]$ und H_D^R (fette Kanten) mit $V(H_D^R) \cap D = S$ 24
3.13 Graph und seine Baumzerlegung 28
3.14 Trick mit künstlicher Kante cd 28
3.15 Durchschnittsmengen von X mit dem Elter Y und dem Kind W 41

4.1 Baumweite tw und Güte r aller 281 Instanzen 52
4.2 Baumweite tw und Güte r der einzelnen Testsets 53
4.3 Knotenanzahl $|V|$ und Güte r aller 281 Instanzen 54
4.4 Knotenanzahl $|V|$ und Güte r der einzelnen Testsets 55

4.5 Terminal-Dichte $\frac{|T|}{|V|}$ und Güte r aller 281 Instanzen 56
4.6 Terminal-Dichte $\frac{|T|}{|V|}$ und Güte r der einzelnen Testsets 57

5.1 Verbesserter Trick mit künstlicher Kante cd 60

Literaturverzeichnis

[1] *Description of the STP Data Format*, 2001. http://steinlib.zib.de/format.php.

[2] *SteinLib Testsets*, 2015. http://steinlib.zib.de/testset.php.

[3] Adler, I.: *Vorlesungsskriptum: Baumzerlegungen, Algorithmen und Logik.* Goethe-Universität Frankfurt, 2011/12.

[4] Arnborg, S., Corneil D. und Proskurowski A.: *Complexity of finding embeddings in a k-tree.* SIAM J. Algebraic Discrete Meth., 8:277–284, 1987.

[5] Chimani, M., Mutzel P. und Zey B.: *Improved Steiner tree algorithms for bounded treewidth.* J. Discrete Algorithms, 16:67–78, 2012.

[6] Cook, W., Cunningham W., Pulleyblank W. und Schrijver A.: *Combinatorial Optimization.* A Wiley-Intercience Publication, 1998.

[7] Fredman, M. L., Komlós J. und Szemerédi E.: *Storing a Sparse Table with 0(1) Worst Case Access Time.* J. ACM, 31(3):538–544, Juni 1984.

[8] Henze, N.: *Stochastik für Einsteiger.* Vieweg, 2008.

[9] Korach, E. und Solel N.: *Linear time algorithm for minimum weight Steiner tree in graphs with bounded treewidth.* Technical Report 632, Israel Institute of Technology, 1990.

[10] Kurose, F. und Ross W.: *Computernetze.* Pearson, 2002.

[11] Meila, M.: *Vorlesungsskriptum: Statistical Learning: Modeling, Prediction and Computing.* University of Washington, 2011.

[12] Mutzel, P.: *Vorlesungsfolien, Algorithmen & Datenstrukturen.* TU Dortmund, 2012/13.

[13] Mutzel, P.: *Vorlesungsfolien, Graphenalgorithmen.* TU Dortmund, 2013.

[14] Rose, D. J., Tarjan R. E. und Lueker G. S.: *Algorithmic Aspects of Vertex Elimination on Graphs.* SIAM Journal on Computing, 5(2):266–283, 1976.

[15] STEGER, A.: *Vorlesungsskriptum: Graphenalgorithmen.* ETH Zuerich, 2006.

[16] STEIN, W. und JOYNER D.: *SAGE: System for Algebra and Geometry Experimentation.* ACM SIGSAM Bulletin, 39:61–64, 2005.

[17] WEGENER, I.: *Vorlesungsskriptum: Datenstrukturen Algorithmen und Programmierung 2.* TU Dortmund, 2007.

[18] WEI-KLEINER, F.: *Tree Decomposition based Steiner Tree Computation over Large Graphs.* CoRR, arXiv:1305.5757 [cs.DS].

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

Dortmund, den 12. Mai 2015

Adalat Jabrayilov