Jack-III

Ein Branch & Cut–Verfahren zur Lösung
des gewichteten Steinerbaumproblems
in Graphen

Diplomarbeit
angefertigt von
Thorsten Koch

eingereicht beim Fachbereich Mathematik
der Technischen Universität Berlin
im September 1995

korrigierte Fassung
Dezember 1995

Erstgutachter: Prof. Dr. M. Grötschel
Zweitgutachter: Prof. Dr. G. M. Ziegler

Zusammenfassung

In der vorliegenden Arbeit werden die grundlegenden Gedanken und die Implementierung eines Branch & Cut-Algorithmus zur Lösung des gewichteten Steinerbaumproblems in Graphen beschrieben.
Der Algorithmus basiert auf der linearen Relaxierung einer bidirektionalen ganzzahligen Formulierung des Problems. Wir werden das Problem einführen, drei ganzzahlige Modellierungen vorstellen, auf Reduktionsverfahren und Heuristiken eingehen sowie das Verfahren und seine Implementierung darstellen.
Am Ende werden wir die Implementierung an 191 bekannten Probleminstanzen testen und auch optimale Lösungen für zwei nach Wissen des Autors bis zu diesem Zeitpunkt ungelöste Instanzen liefern.

Inhaltsverzeichnis

Kapitel 0

Grundlagen

Hier wollen wir die Bedeutung einiger grundlegender Begriffe und Bezeichnungen klären, die wir in dieser Arbeit verwenden.
Die nachfolgenden Definitionen sind im wesentlichen den „*Ergänzungsblättern zur Vorlesung Kombinatorische Optimierung*“ von M. Grötschel und aus [Mar92] entnommen.

Graphen

Ein (*ungerichteter*) *Graph* G ist ein Tripel (V, E, Ψ), bestehend aus einer nicht-leeren Menge V, einer Menge E und einer *Inzidenzfunktion* $\Psi : E \to V^{(2)}$. Hierbei bezeichnet $V^{(2)}$ die Menge der ungeordneten Paare von (nicht notwendigerweise verschiedenen) Elementen von V. Ein Element aus V heißt *Knoten* (Vertex), ein Element aus E heißt *Kante* (Edge). Zu jeder Kante $e \in E$ gibt es also Knoten $u, v \in V$ mit $\Psi(e) = [u, v] = [v, u]$.
Ein Graph heißt *endlich*, falls V und E endlich sind, andernfalls heißt G *unendlich*. Da wir uns in dieser Arbeit nur mit endlichen Graphen beschäftigen, werden wir nur „Graph“ schreiben, wenn wir „endlicher Graph“ meinen.

Gilt $\Psi(e) = [u, v]$ für eine Kante $e \in E$, dann heißen die Knoten $u, v \in V$ *Endknoten* von e, und wir sagen, daß u und v *inzident* zu e sind oder *auf* e *liegen*, daß e die Knoten u und v *verbindet* und daß u und v *Nachbarn* bzw. *adjazent* sind.

Ein Graph heißt *vollständig*, falls $[u, v] \in E$ für alle $u, v \in V$ mit $u \neq v$. Eine Kante e mit $\Psi(e) = [v, v]$ heißt *Schlinge*, und Kanten $e, f \in E$ mit $\Psi(e) = \Psi(f)$ heißen *parallel*. Einen Graphen, der weder Schlingen noch parallele Kanten enthält, nennen wir *einfach*.

Da wir im weiteren Verlauf nur einfache Graphen betrachten, wollen wir zur Vereinfachung der Notation auf die Inzidenzfunktion verzichten und schreiben $e = [u, v]$ mit $e \in E, u \neq v$ und $u, v \in V$. Wir bezeichnen dann mit $G = (V, E)$ einen endlichen, einfachen, ungerichteten Graphen.

Für eine Kantenmenge $F \subseteq E$ bezeichnet $V(F)$ die Menge aller Knoten, die zu einer Kante in F inzident sind, und für $W \subseteq V$ bezeichnet $E(W)$ die Menge aller Kanten, für die beide Endknoten in W sind.

Eine Menge F von Kanten heißt (ungerichteter) *Schnitt*, wenn es eine Knotenmenge $W \subseteq V$ gibt, so daß $F = \delta(W) := \{[u, v] \in E \mid u \in W \text{ und } v \in V \setminus W\}$ gilt. Wir nennen $\delta(W)$ den durch W *induzierten* Schnitt. Wir schreiben statt $\delta(\{v\})$ kurz δ_v. Wir nennen eine Menge von

nicht-leeren Knotenmengen $P = \{V_1, \ldots, V_{|P|}\}$ eine *Partition* von V, wenn die Vereinigung der Knotenmengen V ergibt und die Elemente von P paarweise disjunkt sind.

Der *Grad* eines Knotens $v \in V$ ist die Anzahl der zu ihm inzidenten Kanten. Bei einfachen Graphen ist der Grad gleich $|\delta_v|$.

Betrachten wir mehrere Graphen gleichzeitig, so indizieren wir die benötigten Symbole mit der Bezeichnung des Graphen.

Ein Graph $H = (W, F)$ heißt *Subgraph* eines Graphen $G = (V, E)$, falls $W \subseteq V$ und $F \subseteq E$ gilt. Wir bezeichnen den Subgraphen $(W, E(W))$ von G als von W induziert.

Für einen Graphen $G = (V, E)$ und eine Knotenmenge $W \subseteq V$ bezeichnen wir mit $G(W)$ den Graphen, der durch *Kontraktion* der Knotenmenge W entsteht. Das heißt, die Knotenmenge von $G(W)$ besteht aus den Knoten $V \setminus W$ und einem neuen Knoten w, der an die Stelle der Knotenmenge W tritt. Die Kantenmenge von $G(W)$ enthält alle Kanten aus $E(V \setminus W)$ und alle Kanten aus $\delta(W)$, wobei der Endknoten aus W durch w ersetzt wird.

Digraphen

Ein *gerichteter Graph* oder *Digraph* $D = (V, A, \Psi)$ besteht aus einer (endlichen) nicht-leeren Knotenmenge V, einer (endlichen) Menge A von *Bögen* (Arc) und einer Inzidenzfunktion $\Psi : A \to V \times V$. Wir wollen auch hier auf die Verwendung der Inzidenzfunktion verzichten. Ein Bogen a ist dann ein geordnetes Paar von Knoten, also $a = (u, v)$ mit $u, v \in V$, und wir bezeichnen u als *Anfangs-* oder *Startknoten* und v als *End-* oder *Zielknoten*. u heißt dann *Vorgänger* von v, v Nachfolger von u und a ist *inzident* zu u und v.

Wie bei den ungerichteten Graphen sei ein einfacher Digraph frei von parallelen Bögen und Schlingen. Also bezeichne $D = (V, A)$ einen endlichen, einfachen, gerichteten Graphen.
Wir nennen zwei Bögen $a = (u, v)$ und $\bar{a} = (v, u)$ *antiparallel* und bezeichnen $\bar{a}$ als *Gegenbogen* zu a.

Es besteht die Möglichkeit, mittels einer Abbildung $B : (V, V^{(2)}) \to (V, V \times V)$ einen ungerichteten Graphen $G = (V, E)$ in einen gerichteten Graphen $D = (V, A)$ zu transformieren. Dabei werden zu jeder Kante $[u, v] \in E$ zwei Bögen (u, v) und $(v, u) \in A$ erzeugt. Wir bezeichnen einen Digraphen $D = (V, A) = B(V, E)$ als *bidirektional* und werden statt $B(G)$ auch kurz D_G schreiben.

Für $W \subseteq V$ sei $\delta^+(W) := \{(i, j) \in A \mid i \in W \text{ und } j \notin W\}$, $\delta^-(W) := \delta^+(V \setminus W)$ und $\delta(W) := \delta^+(W) \cup \delta^-(W)$. Die Bogenmenge $\delta^+(W)$ (bzw. $\delta^-(W)$) heißt *Schnitt*. Ist $s \in W$ und $t \notin W$, so heißt $\delta^+(W)$ auch (s, t)-Schnitt. Wir nennen $\delta^+(W)$ den durch W *induzierten* Schnitt. Wir schreiben statt $\delta^+(\{v\})$ und $\delta^-(\{v\})$ kurz δ_v^+ und δ_v^-.

Ketten, Wege, Kreise, Bäume

In einem Graphen oder Digraphen heißt eine endliche Folge $W = (v_0, e_1, v_1, e_2, v_2, \ldots, e_k, v_k)$, $k \geq 0$, die mit einem Knoten beginnt und endet und in der Knoten und Kanten (Bögen) alternierend auftreten, so daß jede Kante (jeder Bogen) e_i mit den beiden Knoten v_{i-1} und v_i inzident ist, eine *Kette*. Der Knoten v_0 heißt *Anfangsknoten*, v_k *Endknoten* der Kette und die Knoten $v_1, \ldots, v_{k-1}$ heißen *innere Knoten*. W wird auch $[v_0, v_k]$-Kette genannt. Die Zahl

k heißt *Länge* der Kette. Falls in einem Digraphen alle Bögen e_i der Kette W gleichgerichtet, also von der Form (v_{i-1}, v_i) sind, so nennt man W *gerichtete Kette* bzw. (v_0, v_k)-Kette.

Eine Kette, in der alle Knoten (und infolgedessen auch Kanten bzw. Bögen) verschieden sind, heißt *Weg*. Eine Kette in der alle Kanten bzw. Bögen verschieden sind, heißt *Pfad*. Ein Weg oder Pfad in einem Digraphen, der eine gerichtete Kette ist, heißt *gerichteter Weg* oder *gerichteter Pfad*. Wie bei Ketten sprechen wir von $[u, v]$-Wegen und (u, v)-Wegen.

Eine Kette heißt *geschlossen*, wenn ihre Länge positiv ist und Anfangs- und Endknoten übereinstimmen. Eine geschlossene (gerichtete) Kette, in der der Anfangsknoten und alle inneren Knoten voneinander verschieden sind, heißt *Kreis* (*gerichteter Kreis*). Die Anzahl der Kanten (Bögen) eines Kreises ist seine *Länge*.

Ein Graph G heißt *zusammenhängend*, falls es zu jedem Paar von Knoten s, t einen $[s, t]$-Weg in G gibt. Ein Digraph D heißt stark zusammenhängend, falls es zu je zwei Knoten s, t sowohl einen gerichteten (s, t)- als auch einen (t, s)-Weg in D gibt. Die *Komponenten* (*starken Komponenten*) eines Graphen (Digraphen) sind die bezüglich Kanteninklusion (Bogeninklusion) maximalen zusammenhängenden Untergraphen von G (maximalen stark zusammenhängenden Unterdigraphen von D).

Eine Kantenmenge F eines Graphen $G = (V, E)$ heißt *trennend*, falls $G' = (V, E \setminus F)$ unzusammenhängend ist. Für Graphen G, die mehr als nur einen Knoten enthalten, bezeichnen wir $\lambda(G) := \min\{|F| \mid F \subseteq E \text{ trennend }\}$ als die *Kantenzusammenhangszahl*. Für Graphen G mit nur einem Knoten setzen wir $\lambda(G) = 0$. Falls $\lambda(G) \geq k$, so nennen wir G *k-fach kantenzusammenhängend*. Im weiteren werden wir einfach k-fach zusammenhängend schreiben, wenn k-fach kantenzusammenhängend gemeint ist.
Analog bezeichnen wir einen Digraphen $D = (V, A)$ als stark *k-fach bogenzusammenhängend*, falls alle Paare $s, t \in V$ mit $s \neq t$ durch mindestens k bogendisjunkte (s, t)-Wege verbunden sind. Auch in diesem Fall werden wir den Digraphen einfach als k-fach zusammenhängend bezeichnen.

Sei $G = (V, E)$ ein Graph. Eine Knotenmenge $W \subseteq V$ heißt *trennend*, falls $(V \setminus W, E(V \setminus W))$ unzusammenhängend ist. Für einfache Graphen $G = (V, E)$ sei die *Zusammenhangszahl* $\kappa(G) := \min\{|W| \mid W \subseteq V \text{ ist trennend}\}$. Falls $\kappa(G) \geq k$, nennen wir G *k-fach knotenzusammenhängend*.

Eine Kante e von $G = (V, E)$ heißt *Brücke*, falls $(V, E \setminus \{e\})$ mehr Komponenten als G hat und ein Knoten v von G heißt *Artikulation*, falls $(V \setminus \{v\}, E)$ mehr Komponenten als G hat.

Ein *Wald* ist ein Graph, der keinen Kreis enthält. Ein zusammenhängender Wald heißt *Baum*. Eine *Verzweigung* (Branching) B ist ein Digraph, der ein Wald ist, so daß jeder Knoten aus B Zielknoten von höchstens einem Bogen von B ist. Eine zusammenhängende Verzweigung heißt *Arboreszenz*. Eine Arboreszenz enthält einen besonderen Knoten r, genannt Wurzel mit $\delta_r^- = \emptyset$, von dem aus jeder andere Knoten auf genau einem gerichteten Weg erreicht werden kann. Wir bezeichnen einen Knoten v eines Baumes oder einer Arboreszenz für den $|\delta_v| = 1$ gilt, als *Blatt*. Wir sagen, daß ein Baum (bzw. eine Arboreszenz) $H = (W, F)$ die Knoten in $W \subseteq V$ *aufspannt*.

Wir nennen einen Graphen G (Digraphen D) *gewichtet*, wenn jeder Kante (jedem Bogen)

ein *Kantengewicht*[1] $c_e \in \mathbb{R}_+$ zugeordnet ist.
Für einen gewichteten Graphen $G = (V, E)$ und Kantengewichte $c_e \in \mathbb{R}_+$ bezeichnen wir einen Baum $H^* = (V, F^*)$ als *gewichtsminimal aufspannenden Baum* oder *minimal spannenden Baum* bezüglich c, wenn es keinen Baum $H = (V, F)$ in G gibt, der V aufspannt und für den $\sum_{e \in F} c_e < \sum_{e \in F^*} c_e$ gilt. Es sein dann $\text{mst}(G, c) \subseteq E$ die Kantenmenge eines minimal spannenden Baumes in G.
Sei $D_G = (V, A)$ der von einem gewichteten Graphen G mit Kantengewichten $c_e \in \mathbb{R}_+$ induzierte bidirektionale Graph, dann seien mit $c_{(u,v)} = c_{(v,u)} := c_{[u,v]}$, $u, v \in V$, Bogengewichte für D_G gegeben.

Lineare Algebra

Wir bezeichnen mit $\mathbb{N}$ die Menge der natürlichen Zahlen und mit $\mathbb{N}_0$ die Menge $\mathbb{N} \cup \{0\}$. $\mathbb{R}$ bezeichnet die reellen Zahlen und $\mathbb{R}_+$ die reellen nicht-negativen Zahlen.

Wir fassen einen Vektor x als Spaltenvektor auf und bezeichnen mit dem oberen Index „T“ die Transposition. Falls wir die Dimension von Matrizen oder Vektoren nicht explizit spezifizieren, gehen wir davon aus, daß die Dimensionen verträglich sind.

Für eine Teilmenge $S \subseteq \mathbb{R}^n$, $S \neq \emptyset$, bezeichnen wir mit $\text{conv}(S)$ die konvexe und mit $\text{cone}(S)$ die konische Hülle von S. Die Dimension von S, in Zeichen $\dim(S)$, ist die maximale Anzahl affin unabhängiger Vektoren aus S minus eins. Ist $\dim(S) = n$, so heißt S *volldimensional*. Mit $\text{diff}(S) := \{x - y \mid x, y \in S\}$ bezeichnen wir die *Differenzmenge* von S.

Für eine endliche Menge E bezeichnen wir mit $\mathbb{R}^E$ die Menge der Abbildungen von E nach $\mathbb{R}$. Es ist zweckmäßig die Elemente aus $\mathbb{R}^E$ als Vektoren mit $|E|$ Komponenten zu betrachten, wobei jede Komponente eines Vektors $x \in \mathbb{R}^E$ mit einem Element aus E indiziert wird, das heißt $x = (x_e)_{e \in E}$. Für eine Teilmenge $F \subseteq E$ definieren wir den *Inzidenzvektor* $\chi^F \in \mathbb{R}^E$ von F durch $\chi^F_e = 1$, falls $e \in F$, und $\chi^F_e = 0$, falls $e \in E \setminus F$. Umgekehrt definieren wir für jeden 0/1 Vektor $x \in \mathbb{R}^E$ durch $F_x := \{e \in E \mid x_e = 1\}$ die *Indexmenge* von x.

Polyedertheorie

Wir nennen $Ax \leq b$ ein System von linearen Ungleichungen, wenn A eine reelle $m \times n$ Matrix und $b \in \mathbb{R}^m$ ist. Die Lösungsmenge $\{x \in \mathbb{R}^n \mid Ax \leq b\}$ eines solchen Systems heißt *Polyeder*.

Ist $a \in \mathbb{R}^n \setminus \{0\}$ und $\alpha \in \mathbb{R}$, so stellt $\{x \in \mathbb{R}^n \mid a^T x \leq \alpha\}$ einen *Halbraum* und $\{x \in \mathbb{R}^n \mid a^T x = \alpha\}$ eine *Hyperebene* dar. Ein Polyeder ist also der Durchschnitt endlich vieler Halbräume.

Eine Ungleichung $a^T x \leq \alpha$ heißt *gültig* für ein Polyeder P, falls $P \subseteq \{x \in \mathbb{R}^n \mid a^T x \leq \alpha\}$.

Eine Menge $F \subseteq P$ heißt *Seitenfläche* von P, falls es eine Ungleichung $a^T x \leq \alpha$ gibt mit $F = \{x \in P \mid a^T x = \alpha\}$. Wir sagen auch, F wird von $a^T x \leq \alpha$ induziert.

Ist die Menge $\{v\}$ eine Seitenfläche von P für den Punkt $v \in P$, so nennt man v eine *Ecke* von P.

[1] Die Bezeichnungen *Gewicht*, *Kosten* und *Länge* einer Kante werden synonym gebraucht.

Eine *Facette* von P ist eine bezüglich Mengeninklusion maximale Seitenfläche von P mit $\emptyset \neq F \neq P$. Wir nennen eine Ungleichung $a^T x \leq \alpha$, eine *facettendefinierende* oder *facetteninduzierende* Ungleichung von P, wenn $F = \{x \in P \mid a^T x = \alpha\}$ eine Facette von P ist.

Für die von einer Ungleichung $a^T \leq \alpha$ induzierte Seitenfläche F eines Polyeders P sind folgende Aussagen äquivalent:

1. F ist eine Facette von P.
2. $\dim(F) = \dim(P) - 1$.
3. Ist P volldimensional und $\bar{a}^T x \leq \bar{\alpha}$ eine gültige Ungleichung für P mit $F \subseteq \{x \in P \mid \bar{a}^T x = \bar{\alpha}\}$, dann existiert ein $\lambda > 0$ mit $a = \lambda\bar{a}$ und $\alpha = \lambda\bar{\alpha}$.

Es gibt auch noch eine andere Darstellungsform für Polyeder unter Verwendung der Operatoren für konvexe und konische Hüllen. Jedes Polyeder $P \subseteq \mathbb{R}^n$ läßt sich beschreiben durch

$$P = \text{conv}(V) + \text{cone}(E),$$

wobei V und E endliche Mengen aus $\mathbb{R}^n$ sind. Hat P eine Ecke, so ist V die Menge der Ecken von P. Ist $E = \emptyset$, so nennt man P auch ein *Polytop*. Ein nicht-leeres Polytop läßt sich also auch als die konvexe Hülle seiner Ecken darstellen.

Lineare Programmierung

Das Problem, eine lineare Funktion $c^T x$ über einem Polyeder $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ zu minimieren (maximieren), heißt *lineares Programmierungsproblem* oder kurz *lineares Programm* oder „LP“. Lineare Programme werden häufig in der Form

$$\begin{array}{c} \min c^T x \\ Ax \leq b \end{array} \quad \text{bzw.} \quad \begin{array}{c} \max c^T x \\ Ax \leq b \end{array}$$

geschrieben. Ein Vektor $\bar{x}$ mit $A\bar{x} \leq b$ heißt *zulässige Lösung* des linearen Programms. Gilt für eine zulässige Lösung x^*, daß $c^T x^* \leq c^T x$ (bzw. $c^T x^* \geq c^T x$) für alle zulässigen Lösungen x, dann heißt x^* *Optimallösung*.

Die lineare Funktion $c^T x$ heißt *Zielfunktion* des linearen Programms. Per Definition ist die Menge der Optimallösungen eine Seitenfläche des Polyeders. Falls ein Polyeder P eine Ecke hat, so hat auch jede nicht-leere Seitenfläche eine Ecke. Hat also P eine Ecke und ist das lineare Programm $\min c^T x,\ x \in P$ (bzw. $\max c^T x,\ x \in P$) beschränkt, so gibt es eine Optimallösung x^*, die eine Ecke von P ist. Insbesondere hat jedes nicht-leere Polytop immer eine optimale Ecklösung.

Das Problem, eine lineare Zielfunktion $c^T x$ über den ganzzahligen Vektoren eines Polyeders $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ zu minimieren (maximieren), heißt *ganzzahliges lineares Programm* oder kurz „GP“. Fordert man darüber hinaus, daß $x_i \in \{0, 1\}$, so erhält man ein *binäres lineares Programmierungsproblem* oder „BP“. Dasjenige LP, das sich aus einem ganzzahligen oder binären Programm durch Weglassen der Ganzzahligkeitsbedingungen ergibt, nennt man *LP-Relaxierung* des GP bzw. BP. Wenn wir nachfolgend ein LP, GP oder BP betrachten, so ist immer die Minimierung bezüglich der Zielfunktion gemeint.

Notation	
V	eine endliche Menge von Knoten.
$V(F)$	$\subseteq V$, $F \subseteq E$, die Menge aller Knoten, die zu einer Kante in F inzident sind.
T	$\subseteq V$, eine ausgezeichnete Teilmenge von V, die *Terminale*.
N	$= V \setminus T$, die Menge aller Nicht-Terminale.
E	$\subseteq \{[a,b] \mid a,b \in V\}$, eine Teilmenge der ungeordneten Paare von Knoten, die Menge der Kanten.
A	$\subseteq \{(a,b) \mid a,b \in V\}$, eine Teilmenge der geordneten Paare von Knoten, die Menge der Bögen.
$E(W)$	$\subseteq E$, $W \subseteq V$, die Menge aller Kanten zwischen zwei Knoten in W.
G	$= (V,E)$ ein einfacher ungerichteter Graph, bestehend aus der Knotenmenge V und der Kantenmenge E.
$G(W)$	Der Graph, der durch die Kontraktion der Knotenmenge $W \subseteq V$ entsteht.
$B(V,E)$	$= (V,A)$ ein bidirektionaler Digraph, bestehend aus der Knotenmenge V und der Bogenmenge A.
D_G	$= B(G)$ der durch den Graphen G induzierte bidirektionale Digraph.
$\mathrm{ST}(G,T,c)$	das durch den Graphen $G = (V,E)$, die Terminalmenge T und den Kostenvektor $c \in \mathbb{R}_+^{\lvert E\rvert}$ gegebene Steinerbaumproblem.
$\mathrm{ST}(D_G,T,c)$	das durch den bidirektionalen Digraphen $D_G = (V,A) = B(G)$, die Terminalmenge T und den Kostenvektor $c \in \mathbb{R}_+^{\lvert A\rvert}$ gegebene gerichtete Steinerbaumproblem.
S	$\subseteq E$, eine (zulässige) Lösung eines (bidirektionalen) Steinerbaumproblems $\mathrm{ST}(V,E,T)$.
χ^S	$\in \{0,1\}^{\lvert E\rvert}$, der einer Lösung S zugehörige Inzidenzvektor mit $\chi_e^S = 1$, falls $e \in S$, und $\chi_e^S = 0$, falls $e \in E \setminus S$.
δ_v	$= \{[a,b] \in E \mid a = v \text{ oder } b = v,\ v \in V\}$, die Menge aller mit dem Knoten v verbundenen Kanten.
δ_v^-	$= \{(a,b) \in A \mid b = v,\ v \in V\}$, die Menge aller zum Knoten v hinführenden Bögen.
δ_v^+	$= \{(a,b) \in A \mid a = v,\ v \in V\}$, die Menge aller vom Knoten v wegführenden Bögen.
$\delta(W)$	$= \{[a,b]) \in E \mid (a \in W \text{ und } b \notin W) \text{ oder } (a \notin W \text{ und } b \in W),\ W \subseteq V\}$, die Menge aller Kanten zwischen W und $V \setminus W$.
$\lvert x\rvert$	die Kardinalität der Menge x.
c_e	$\in \mathbb{R}_+$, die einer Kante $e \in E$ zugeordneten Kosten.
$c(E)$	$= \sum_{e \in E} c_e$.
u_e	$\in \mathbb{N}$, die Kapazität einer Kante $e \in E$.
$d(u,v)$	$u,v \in V$, die Länge eines kürzesten Weges von u nach v.
$\mathrm{mst}(G,c)$	$= \mathrm{argmin}_{F \subseteq E}\{c(F) \mid V(F) = V\}$, die Kantenmenge eines Baumes, der V gewichtsminimal aufspannt. $c \in \mathbb{R}_+^{\lvert E\rvert}$ ist der Vektor der Kantengewichte.
•	stellt in den abgebildeten Graphen ein Nicht-Terminal dar.
■	stellt in den abgebildeten Graphen ein Terminal dar.

Kapitel 1

Das Problem

In dieser Arbeit wollen wir uns mit dem *gewichteten Steinerbaumproblem* in Graphen befassen:

> Gegeben sei ein Graph $G = (V, E)$ und eine ausgezeichnete Menge $T \subseteq V$ von Knoten sowie Kantengewichte $c_e \geq 0$ für alle $e \in E$:
> Finde einen Baum minimalen Gewichts in G, der T aufspannt.

Dabei wird T als die Menge der *Terminalen*[1] bezeichnet. Ein Steinerbaum S ist ein Baum in G, der mindestens T aufspannt. Die im Baum enthaltenen Knoten aus $V \setminus T$ bezeichnet man als *Steinerknoten*.

Bereits im 17. Jahrhundert wurde erstmals das Problem, in der euklidischen Ebene drei Punkte mit einem Netzwerk minimaler Länge zu verbinden, erwähnt.
Der Name „Steiner“ nach Jacob Steiner (1796-1863) wurde dem Problem, [HR92] zufolge, von Courant und Robbins im Jahre 1941 in ihrem Buch *What is Mathematics ?* zugeschrieben.

Das Steinerbaumproblem in Graphen ist ein seit vielen Jahren untersuchtes Problem, und so sind schon viele gängige Optimierungsverfahren darauf angewendet worden.
Als exakte Verfahren gibt es diverse Arten von Enumerationen [Hak71] und dynamischer Programmierung [DW71], zur Berechnung unterer Schranken Lagrange Relaxierung [Bea84], [Bea89], [Luc93], Dual Ascent [Won84] sowie Schnittebenenverfahren [Ane80], [CGR92], [Luc93].
Darüber hinaus sind eine Anzahl heuristischer Verfahren bekannt, die meist sehr gute Lösungen (nicht mehr als 10 % vom Optimum entfernt) liefern.

R. M. Karp hat in [Kar72] gezeigt, daß das Steinerbaumproblem für beliebige Graphen $\mathcal{NP}$-schwer ist. Falls aber $|T| = 2$, ist das Problem äquivalent der Bestimmung eines kürzesten Weges zwischen den beiden Terminalen. Wenn $T = V$, dann ist das Problem äquivalent der Berechnung eines minimal spannenden Baumes für G und damit polynomial lösbar.

Obgleich das Problem $\mathcal{NP}$-schwer ist, folgt daraus nicht, daß es für größere Probleminstanzen[2] unlösbar ist. Wie unter anderen von Chopra, Gorres und Rao in [CGR92] oder von

[1] Der Begriff stammt vermutlich aus dem Schaltungsdesign, wo Steinerbaumprobleme häufig anzutreffen sind. Er ist auch insofern treffend, als bei einem minimalen Steinerbaum alle Blätter Terminale sind.

[2] Wir bezeichnen jedes konkret vorliegende Steinerbaumproblem eine *Instanz* (Manifestation) des allgemeinen Steinerbaumproblems.

Lucena in [Luc93] gezeigt wurde, können Steinerbaumprobleme in Graphen mit mehreren tausend Knoten und zehntausenden von Kanten durchaus gelöst werden.

Wir werden in dieser Arbeit versuchen, die Grenze des Machbaren wieder ein kleines Stück weiter zu schieben und die Implementierung eines Branch & Cut–Verfahrens beschreiben, das eine große Zahl bekannter Probleminstanzen löst und auch für zwei nach Wissen des Autors bisher ungelöste Instanzen eine optimale Lösung ermittelt.

Bevor wir uns aber fragen, *wie* wir das Problem lösen, müssen wir festlegen, *was* eine Lösung ist:

Eine *Lösung* eines Steinerbaumproblems $\mathrm{ST}(G, T, c)$ sei eine Menge $S \subseteq E$ von Kanten. Wir erhalten das Gewicht (oder die Kosten) $c(S)$ der Lösung, indem wir die Gewichte der benutzten Kanten aufsummieren. Also

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

Wir wollen eine Lösung S *zulässig* nennen, wenn T von den in S enthaltenen Kanten aufgespannt wird, also $T \subseteq V(S)$ gilt.

Wir bezeichnen eine Lösung als *minimal* oder *kantenminimal*, wenn sie zulässig ist und der durch S induzierte Subgraph von G einen Baum bildet, dessen Blätter Terminale sind[3].

Eine *optimale* oder *gewichtsminimale* Lösung S^* sei eine zulässige kantenminimale Lösung minimalen Gewichts.

Wir gehen (zumindest in den theoretischen Betrachtungen) dabei immer davon aus, daß G zweifach zusammenhängend ist.

Andernfalls ist G entweder unzusammenhängend, dann können wir die jeweils zusammenhängenden Komponenten als eigenständige Probleme betrachten[4], oder G ist nur einfach zusammenhängend, dann gibt es entweder (i) eine *Steinerbrücke*, d.h. eine Kante $e_b \in E$, die in jedem Steinerbaum enthalten sein muß, oder (ii) einen Artikulationsknoten k_b.

(i) Im Falle einer Steinerbrücke zerlegt man G durch Entfernen von $e_b = [u, v]$ in zwei disjunkte in sich zusammenhängende Graphen $G_i = (V_i, E(V_i))$ für $i = 1, 2$ und macht die Endpunkte von e_b zu Terminalen. Hat man die optimalen Lösungen S_i von $\mathrm{ST}(G_i, V_i \cap (T \cup \{u, v\}), c)$ für $i = 1, 2$ ermittelt, so erhält man mit $S = S_1 \cup S_2 \cup \{e_b\}$ eine gewichtsminimale Lösung von $\mathrm{ST}(G, T, c)$.

(ii) In diesem Fall kann G in $G_i = (V_i, E(V_i)) \subset G$ mit $\bigcap V_i = \{k_b\}$ und $\bigcup G_i = G$ für $i = 1, \ldots, n$, $n \geq 2$ aufgeteilt werden. Ermittelt man dann die optimalen Lösungen S_i von $\mathrm{ST}(G_i, V_i \cap T \cup \{k_b\}, c)$, so erhält man mit $S = \bigcup_i S_i$ eine gewichtsminimale Lösung von $\mathrm{ST}(G, T, c)$.

[3] Also ist der durch S induzierte Subgraph insbesondere kreisfrei. Ob Kreisfreiheit für eine Lösung gefordert wird, ist nicht entscheidend, da jede Optimallösung diese Eigenschaft offensichtlich hat.

[4] Teilgraphen oder Komponenten ohne Terminal haben, die triviale Lösung $\emptyset$.

Kapitel 2

Modellierungen und polyedrische Betrachtungen

2.1 Modellierungen des Steinerbaumproblems

Um ein Verfahren zur Lösung des Problems entwickeln zu können, müssen wir eine Modellierung wählen, innerhalb derer unser Verfahren arbeiten soll. Dazu wollen wir drei mögliche Formulierungen des Steinerbaumproblems als binären Programms (BP) und deren LP-Relaxierungen vorstellen[1]:

Ein ungerichtetes Modell

In Kapitel 1 haben wir eine Lösung des Steinerbaumproblems $\mathrm{ST}(G,T,c)$ als Kantenmenge definiert. Es liegt daher nahe, für jede Kante des ungerichteten gewichteten Graphen $G(V,E)$ mit Kantengewichten c_e, für alle $e \in E$, eine binäre Variable x_e einzuführen, die den Wert 1 genau dann annimmt, wenn die zugehörige Kante in einer Lösung S enthalten ist.
Das Problem läßt sich dann als binäres Programm $\mathrm{BP}(\mathrm{ST}(G,T,c))$ formulieren[2]:

$$\begin{aligned} \min & \sum_{e\in E} x_e c_e & & \\ \text{s.t.:} & \sum_{e\in\delta(W)} x_e & \geq & \ 1, \text{ für alle } W \subset V,\ W\cap T \neq \emptyset,\ (V\setminus W)\cap T \neq \emptyset \qquad (2.1) \\ & x_e & \in & \ \{0,1\}, \text{ für alle } e \in E. \qquad (2.2) \end{aligned}$$

Man bezeichnet jede Ungleichung in (2.1) auch als *Steinerschnittungleichung*, da eine zulässige Lösung mindestens eine Kante aus jedem Schnitt in G beinhaltet, der jeweils mindestens ein Terminal enthaltende Subgraphen induziert. Es ist also

$$\mathrm{conv}(\{\chi^S \mid S \text{ sei eine zulässige Lösung für } \mathrm{ST}(G,T,c)\})$$

[1] Für eine ausführlichere Übersicht siehe Chopra, Rao [CR94a] und Goemans, Myung [GM93].
[2] Vgl. [Mar92].

das von der konvexen Hülle über den Inzidenzvektoren der Lösungen von $\mathrm{ST}(G,T,c)$ gebildete Polytop äquivalent zu

$$\mathcal{P}_{\mathrm{ST}}(G,T) := \mathrm{conv}(\{x \mid x \text{ erfüllt (2.1) und (2.2) }\})$$

der konvexen Hülle über den Lösungen von $\mathrm{BP}(\mathrm{ST}(G,T,c))$. Lassen wir die Ganzzahligkeitsbedingung (2.2) wegfallen, so erhalten wir als Lösungsmenge der LP-Relaxierung $\mathrm{LP}(\mathrm{ST}(G,T,c))$ von $\mathrm{BP}(\mathrm{ST}(G,T,c))$ das Polytop

$$\mathcal{P}_{\mathrm{LP}}(G,T) := \{x \in [0,1]^{|E|} \mid x \text{ erfüllt (2.1) }\}$$

und es gilt $\mathcal{P}_{\mathrm{ST}}(G,T) \subseteq \mathcal{P}_{\mathrm{LP}}(G,T)$.

Lemma 2.1
Sei $\mathrm{ST}(G,T,c)$ ein Steinerbaumproblem, das keine Steinerbrücke[3] enthält. Dann ist

$$\dim(\mathcal{P}_{\mathrm{ST}}(G,T)) = |E|.$$

Beweis:[4] Es genügt zu zeigen, daß $\lambda^T x = 0$ mit $x \in \mathrm{diff}(\mathcal{P}_{\mathrm{ST}}(G,T)) \Rightarrow \lambda_e = 0$ für alle $e \in E$. Wähle $e \in E$ beliebig. Dann sei S die Kantenmenge eines T aufspannenden Baumes in $(V, E \setminus \{e\})$. Da G keine Steinerbrücke enthält, muß S existieren. Da $T \subseteq V(S)$ gilt, ist S eine zulässige Lösung von $\mathrm{ST}(G,T,c)$, also $\chi^S \in \mathcal{P}_{\mathrm{ST}}(G,T)$, mit $e \notin S$. Da auch $S \cup \{e\}$ eine zulässige Lösung ist, gilt $\chi^{S\cup\{e\}} \in \mathcal{P}_{\mathrm{ST}}(G,T)$ und damit $\lambda_e = \lambda^T(\chi^{S\cup\{e\}} - \chi^S) = 0$. ■

Beispiel 2.2
Man betrachte den in Bild 2.1 gezeigten Graphen $G_{\mathrm{T3}} = (V_{\mathrm{T3}}, E_{\mathrm{T3}})$. Alle Knoten seien Terminale, also $T_{\mathrm{T3}} = V_{\mathrm{T3}}$ und die Kantengewichte seien alle gleich eins. Bild 2.2 stellt dann die Vektoren aller möglichen Lösungen dar. Die ausgefüllten Punkte entsprechen den zulässigen Lösungen. Der in 2.3 dargestellte Einheitswürfel ergibt sich als konvexe Hülle über den Lösungen.

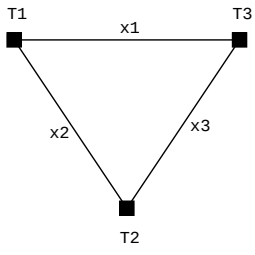


Abbildung 2.1:

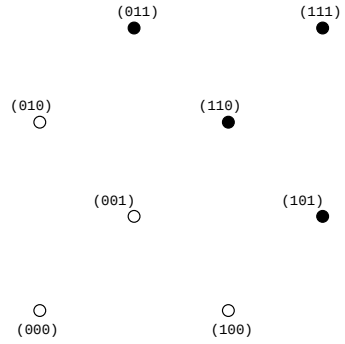


Abbildung 2.2:

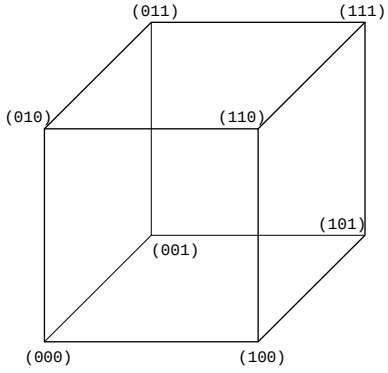


Abbildung 2.3:

Berücksichtigen wir jetzt die Ungleichungen (2.1), erhalten wir das in 2.4 abgebildete Polytop $\mathcal{P}_{\mathrm{ST}}(G,T)$. Zum Vergleich zeigt Bild 2.5 $\mathcal{P}_{\mathrm{LP}}(G,T)$ das durch die LP-Relaxierung definierte

[3] Wir haben auf Seite 8 gesehen, wie Steinerbrücken entfernt werden können.
[4] Vgl. [Mar92] und [GM90]

Polytop. Der Unterschied besteht in der zusätzlich vorhandene Ecke $(0.5, 0.5, 0, 5)$, deren zugehöriger Lösungsvektor zwar alle Ungleichungen (2.1) erfüllt, aber, da nicht ganzzahlig, keine Lösung von $\mathrm{ST}(G, T, c)$ darstellt.
Betrachten wir nur die konvexe Hülle über den kantenminimalen zulässigen Lösungen, so erhalten wir das in Bild 2.6 abgebildete nicht volldimensionale Polytop.

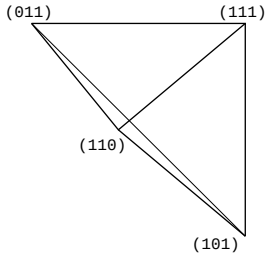


Abbildung 2.4:

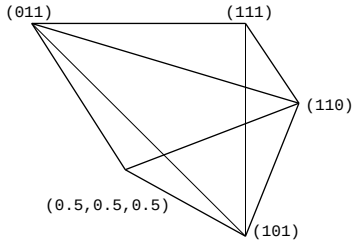


Abbildung 2.5:

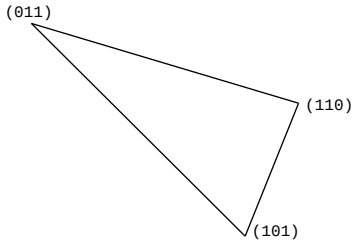


Abbildung 2.6:

Der Vergleich der facetteninduzierenden Ungleichungen von $\mathcal{P}_{\mathrm{ST}}(G_{\mathrm{T3}}, T_{\mathrm{T3}})$ mit $\mathcal{P}_{\mathrm{LP}}(G_{\mathrm{T3}}, T_{\mathrm{T3}})$ zeigt, daß die Steinerschnittungleichungen nicht facetteninduzierend für $\mathcal{P}_{\mathrm{ST}}(G_{\mathrm{T3}}, T_{\mathrm{T3}})$ sind.[5]

$\mathcal{P}_{\mathrm{ST}}(G_{\mathrm{T3}}, T_{\mathrm{T3}})$			$\mathcal{P}_{\mathrm{LP}}(G_{\mathrm{T3}}, T_{\mathrm{T3}})$		
$x_1 + x_2 + x_3$	$\geq$	2	$x_1 + x_2$	$\geq$	1
			$x_1 + x_3$	$\geq$	1
			$x_2 + x_3$	$\geq$	1
x_1	$\leq$	1	x_1	$\leq$	1
x_2	$\leq$	1	x_2	$\leq$	1
x_3	$\leq$	1	x_3	$\leq$	1

Grötschel und Monma haben in [GM90] bewiesen, daß die Ungleichungen $x_e \leq 1$, für $e \in E$, genau dann facetteninduzierend bezüglich $\mathcal{P}_{\mathrm{ST}}(G, T)$ sind, wenn $\mathcal{P}_{\mathrm{ST}}(G, T)$ volldimensional ist. Ebenso ist $x_e \geq 0$ genau dann facettendefinierend, wenn zusätzlich gilt, daß G ohne die Kante e keine Steinerbrücke enthält.

Sei $\mathrm{ST}(G, T, c)$ ein Steinerbaumproblem mit $G = (V, E)$ und $Q = \{W_1, \ldots, W_{|Q|}\}$ eine *Partition* von V, d.h. $W_i \cap W_j = \emptyset$ für $i \neq j$ für alle $i, j \in \{1, \ldots, |Q|\}$ und $\bigcup_{W \in Q} W = V$ und es gelte $W \cap T \neq \emptyset$ für alle $W \in Q$,

dann ist die *Steinerpartitionsungleichung*

$$\sum_{e \in E_Q} x_e \geq |Q| - 1 \quad \text{mit } E_Q := \bigcup_{W \in Q} \delta(W) \tag{2.3}$$

gültig für $\mathcal{P}_{\mathrm{ST}}(G, T)$[6]. Grötschel und Monma zeigen in [GM90], daß es sich bei (2.3) genau dann um eine Klasse von facettendefinierenden Ungleichungen für $\mathcal{P}_{\mathrm{ST}}(G, T)$ handelt, wenn

[5] Die Gleichungssysteme wurden mit PORTA (Polyhedron Representation Transformation Algorithm), einem Programm von T. Christof, M. Stoer und A. Löbel, ermittelt.

[6] Für den Fall $|Q| = 2$ handelt es sich gerade um die Steinerschnitte.

für alle $W \in Q$ der durch $(W, E(W)) \subseteq G$ gebildete Subgraph zusammenhängend und frei von Steinerbrücken bzgl. $W \cap T$ ist und der Graph, der entsteht, wenn jedes $W \in Q$ zu einem Knoten kontrahiert wird, zweifach knotenzusammenhängend ist.

Weitergehende Untersuchungen zu gültigen Ungleichungen und Facetten von $\mathcal{P}_{\mathrm{ST}}(G,T)$ und verwandten Polyedern finden sich in [CR88a], [CR88b], [Goe91] und [GM90].

Ein ungerichtetes Modell mit Knotenvariablen

Hier führen wir zusätzlich Variablen $y_k \in \{0,1\}$ mit $k \in V$ ein, die genau dann eins sind, wenn der Knoten k inzident zu einer in der Lösung S enthaltenen Kante ist[7], also $k \in V(S) \Leftrightarrow y_k = 1$.

Da offensichtlich $y_k = 1$ für alle $k \in T$ gilt, können wir unsere Betrachtung auf die y_k beschränken, für die $k \in N$ ist. Das Steinerbaumproblem stellt sich dann als binäres Programm $\mathrm{BP}'(\mathrm{ST}(G,T,c))$ wie folgt dar:

$$\min \sum_{e \in E} x_e c_e$$

$$\begin{aligned}
\text{s.t.:} \quad \sum_{e \in E} x_e &= \sum_{k \in N} y_k + |T| - 1 && (2.4)\\
\sum_{e \in E(W)} x_e &\leq \sum_{k \in W \setminus T} y_k + |W \cap T| - 1, \text{ für alle } W \subseteq V,\ W \cap T \neq \emptyset && (2.5)\\
\sum_{e \in E(W)} x_e &\leq \sum_{k \in W \setminus \{j\}} y_k, \text{ für alle } j \in W,\ W \subseteq V,\ W \cap T = \emptyset && (2.6)\\
x_e &\in [0,1] && (2.7)\\
y_k &\in \{0,1\}. && (2.8)
\end{aligned}$$

Die Knotenvariablen gehen dabei nicht in das Gewicht der Lösung ein, sondern verfolgen nur, welche Nicht-Terminale von der Lösung erreicht werden, und es gilt

$$x_e \leq y_k \text{ mit } e \in \delta_k, \text{ für alle } k \in N. \tag{2.9}$$

Bei den Ungleichungen (2.5) und (2.6) handelt es sich um sogenannte *verallgemeinerte Kurzzyklusungleichungen*[8]. Die Idee ist jeweils die gleiche wie bei (2.4). Da die optimale Lösung ein aufspannender Baum sein muß, ist das Verhältnis von Knoten zu Kantenanzahl in der Lösung bekannt. Und da wir schon wissen, daß die Terminale in der Lösung enthalten sind, kann keine Nullösung zulässig sein.

Wir erhalten die LP-Relaxierung $\mathrm{LP}'(\mathrm{ST}(G,T,c))$ von $\mathrm{BP}'(\mathrm{ST}(G,T,c))$ und die zugehörige Lösungsmenge $\mathcal{P}'_{\mathrm{LP}}(G,T)$ in gewohnter Weise, indem wir die Bedingungen (2.7) und (2.8) durch $x_e, y_k \in [0,1]$ ersetzen.

Eine weitere gültige Ungleichung ist mit

$$\sum_{e \in \delta_k} x_e \geq 2y_k, \text{ für alle } k \in N \tag{2.10}$$

[7] Vgl. [GM93] und [Luc93].

[8] *Generalized subtour elimination contraints .*

gegeben, da jeder Steinerknoten mindestens den Grad zwei haben muß, um Teil einer minimalen Lösung zu sein.

A. Lucena beschreibt in [Luc93] eine Implementierung dieses Modells in Verbindung mit einer Lagrange-Relaxierung und einem Schnittebenenverfahren. Wir werden die dort erzielten Resultate in Abschnitt 7.3 mit den von uns erzielten vergleichen.

Ein gerichtetes Modell

Benutzen wir einen gewichteten bidirektionalen Graphen $D_G = (V, A) = B(V, E)$ mit Bogengewichten $c_{(u,v)} = c_{(v,u)} := c_{[u,v]}$ für alle $[u, v] \in E$ zur Modellierung, erhalten wir doppelt so viele Variablen wie im ungerichteten Fall.

Für das gerichtete Modell können wir eine Lösung analog zum ungerichteten Modell als Bogenmenge definieren.

Sei $r \in T$ ein beliebiges Terminal. Dann heißt die Lösung S eines bidirektionalen Steinerbaumproblems $\mathrm{ST}(D_G, T, c)$ *zulässig*, wenn es in dem von S induzierten Subgraphen von D_G einen gerichteten Weg von r zu jedem Terminal $t \in T \setminus \{r\}$ gibt[9].

Wir nennen eine zulässige Lösung *bogenminimal*, wenn S eine Arboreszenz induziert, deren Endpunkte Terminale sind[10].

Wir bezeichnen eine Lösung S^* als *gewichtsminimal* oder *optimal*, wenn $(V(S^*), S^*)$ eine T aufspannende Arboreszens minimalen Gewichts ist.
Gesucht wird also eine *Steinerarboreszenz* minimalen Gewichts, und wir können dies als binäres Programm $\mathrm{BP}(\mathrm{ST}(D_G, T, c))$ wie folgendermaßen formulieren:

$$\begin{aligned} \min & \sum_{a \in A} x_a c_a & & \\ \text{s.t.:} & \sum_{a \in \delta^-(W)} x_a & \geq & \ 1, \text{ für alle } W \subseteq V \setminus \{r\},\ W \cap T \neq \emptyset \qquad (2.11) \\ & x_a & \in & \ \{0, 1\}. \qquad (2.12) \end{aligned}$$

Sei nun

$$\mathcal{P}_{\mathrm{ST}}(D_G, T) := \mathrm{conv}(\{\chi^S \mid S \text{ sei eine zulässige Lösung von } \mathrm{ST}(D_G, T, c)\}) \qquad (2.13)$$

das Polytop, das von den zulässigen Lösungen von $\mathrm{BP}(\mathrm{ST}(D_G, T, c))$ aufgespannt wird.

Sei eine Partition $\{V_1, V_2\}$ von V mit $r \in V_1$ und $V_2 \cap T \neq \emptyset$ gegeben, dann nennen wir $C = \{(u, v) \in A \mid u \in V_1 \text{ und } v \in V_2\}$ einen *gerichteten Steinerschnitt.*

Entsprechend bezeichnen wir (2.11) als *gerichtete Schnittungleichungen.* Es sind gültige Ungleichungen bezüglich $\mathcal{P}_{\mathrm{ST}}(D_G, T)$, da es keine zulässige Lösung gibt, die nicht mindestens einen Bogen aus jedem gerichteten Steinerschnitt enthält.

[9] Da der D_G zugrundeliegende Graph G zusammenhängend ist, existiert in D_G immer ein gerichteter Weg von r zu jedem Knoten.

[10] Das heißt nicht, daß alle Terminale auch Endpunkte sein müssen.

Betrachten wir nun

$$\mathcal{P}_{\text{ST}}(D_G, T) := \text{conv}(\{x \mid x \text{ erfüllt (2.11) und (2.12) }\}).$$

$\mathcal{P}_{\text{ST}}(D_G, T)$ ist offenbar äquivalent zu (2.13), da in jeder von einer zulässigen Lösung S induzierten Arboreszenz ein gerichteter Weg von der Wurzel zu jedem anderen Terminal besteht, also $S \cap C \neq \emptyset$ für alle gerichteten Steinerschnitte C in D_G gilt. Andererseits folgt aus $S \cap C \neq \emptyset$ für alle gerichteten Steinerschnitte C in D_G auch, daß die Wurzel mit jedem Terminal verbunden ist.

Wenn wir nun die Ganzzahligkeitsbedingung (2.12) bei BP(ST(D_G, T, c)) fallen lassen, erhalten wir mit dem Polytop

$$\mathcal{P}_{\text{LP}}(D_G, T) := \{x \in [0,1]^{|E|} \mid \sum_{e \in C} x_e \geq 1, \text{ für alle gerichteten Steinerschnitte } C\}$$

die Menge aller Lösungen der LP-Relaxierung LP(ST(D_G, T, c)) von BP(ST(D_G, T, c)) und es gilt $\mathcal{P}_{\text{ST}}(D_G, T) = \text{conv}(\{x \mid x \in \mathcal{P}_{\text{LP}}(D_G, T) \cap \{0,1\}^{|E|}\})$.

Lemma 2.3
Sei $G = (V, E)$ ein zweifach zusammenhängender Graph und $D_G = B(G) = (V, A)$ der durch G induzierte bidirektionale Graph und ST(D_G, T, c) ein Steinerbaumproblem über D_G mit Wurzel $r \in T$. Dann ist

$$\dim(\mathcal{P}_{\text{ST}}(D_G, T)) = |A|.$$

Beweis: Es genügt zu zeigen, daß $\lambda^T x = 0$ mit $x \in \text{diff}(\mathcal{P}_{\text{ST}}(D_G, T)) \Rightarrow \lambda_a = 0$ für alle $a \in A$.
Wähle $a \in A$ beliebig. Dann sei S die Bogenmenge einer V aufspannenden Arboreszenz mit Wurzel r in $(V, A \setminus \{a\})$. Da der D_G zugrundeliegende Graph G zweifach zusammenhängend ist, muß S existieren. Da $T \subseteq V(S)$ gilt, ist S eine zulässige Lösung von ST(D_G, T, c) mit $a \notin S$. Da auch $S \cup \{a\}$ eine zulässige Lösung ist, gilt $\chi^{S \cup \{a\}} \in \mathcal{P}_{\text{ST}}(D_G, T)$ und damit $\lambda_a = \lambda^T(\chi^{S \cup \{a\}} - \chi^S) = 0$. ∎

Um eine Aussage über die gerichteten Steinerschnittungleichungen machen zu können und uns die nachfolgenden Untersuchungen zu erleichtern, erweitern wir $\mathcal{P}_{\text{ST}}(D_G, T)$ noch etwas und definieren

$$\mathcal{P}^+_{\text{ST}}(D_G, T) := \mathcal{P}_{\text{ST}}(D_G, T) + \mathbb{R}^{|A|}_+.$$

Lemma 2.4
Eine Ungleichung der Form

$$\pi^T x \geq 1 \text{ mit } \pi \in \mathbb{R}^{|A|} \tag{2.14}$$

definiert genau dann eine Facette von $\mathcal{P}^+_{\text{ST}}(D_G, T)$, wenn (2.14) gültig ist und wenn wir eine Menge X mit $|X| = |\text{supp}(\pi)|$ von Vektoren aus $\mathcal{P}^+_{\text{ST}}(D_G, T)$ finden können, die (2.14) mit Gleichheit erfüllen und für die gilt, daß alle πx mit $x \in X$ linear unabhängig sind.

Beweis:[11] Für ein $x \in \mathcal{P}^+_{\text{ST}}(D_G, T)$ und ein $\bar{a} \in A$, sei $\bar{x}$ wie folgt definiert: $\bar{x}_a := x_a + 1$ für $a = \bar{a}$ und $\bar{x}_a := x_a$ sonst. Da $\mathcal{P}^+_{\text{ST}}(D_G, T)$ den positiven Orthanten als Rezessionskegel hat,

[11] Vgl. [CR88a].

gilt $\bar{x} \in \mathcal{P}^+_{\text{ST}}(D_G, T)$. Es sei $\bar{A} = \{a \in A \mid \pi_a = 0\}$, dann gilt für alle $\bar{a} \in \bar{A}$, daß aus x erfüllt (2.14) mit Gleichheit gleiches für $\bar{x}$ folgt.
Wir können jetzt für ein beliebiges $x \in X$ und für alle Bögen aus $\bar{A}$ eine Menge $\bar{X}$ von Vektoren $\bar{x}$ erzeugen. Die Menge $X \cup \bar{X}$ enthält dann $|A|$ linear unabhängige Vektoren x für die $\pi x = 1$ gilt. Also ist (2.14) eine Facette für $\mathcal{P}^+_{\text{ST}}(D_G, T)$.

Definiert (2.14) eine Facette, so müssen die $|A|$ geforderten Vektoren existieren. Daraus folgt aber auch, daß es $|\text{supp}(\pi)|$ Vektoren, für die $\pi x = 1$ gilt, geben muß. ∎

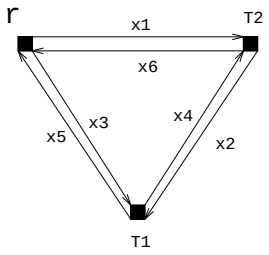
Satz 2.5
Jede durch einen bogenminimalen gerichteten Steinerschnitt C induzierte Ungleichung (2.11) definiert eine Facette des Polytops $\mathcal{P}^+_{\text{ST}}(D_G, T)$, wenn der zugrundeliegende Graph G zusammenhängend ist.

Beweis:[12] Seien V_1 und V_2 die durch einen gerichteten Steinerschnitt induzierten Subgraphen. Da D_G bidirektional und $(V_1, E(V_1))$ zusammenhängend ist, gibt es einen gerichteten Weg von r zu jedem Knoten in $V_1(C)$. Also gibt es eine Arboreszenz mit Bogenmenge S_1, die beginnend bei r, V_1 aufspannt.

Sei $a = (v_i, v_j)$ ein beliebiger Bogen aus C, dann existiert eine Arboreszenz mit Bogenmenge S_2, die beginnend bei v_j, $V_2 \cap T$ aufspannt, wenn V_2 zusammenhängend ist.

$S_e = S_1 \cup S_2 \cup \{e\}$ induziert dann eine Arboreszenz in D_G, und χ^{S_e} erfüllt (2.11) mit Gleichheit. Wir erhalten also insgesamt $|C|$ linear unabhängige Vektoren. Nach Lemma 2.4 ist die Ungleichung also facettendefinierend[13]. ∎

Beispiel 2.6
Hier wollen wir noch einmal den Graphen G_{T3} betrachten. Die bidirektionale Variante $D_{G_{\text{T3}}}$ ist in Abbildung 2.7 zu sehen. Daneben sind die facetteninduzierenden Ungleichungen von $\mathcal{P}_{\text{ST}}(D_{G_{\text{T3}}}, T_{\text{T3}})$ abgebildet. In diesem Fall gilt für $D_{G_{\text{T3}}}$, daß $\mathcal{P}_{\text{ST}}(D_{G_{\text{T3}}}, T_{\text{T3}}) = \mathcal{P}_{\text{LP}}(D_{G_{\text{T3}}}, T_{\text{T3}})$ ist.



$\mathcal{P}_{\text{ST}}(D_{G_{\text{T3}}}, T_{\text{T3}})$

$$
\begin{array}{rcl}
x_1 + x_3 & \geq & 1 \\
x_1 + x_4 & \geq & 1 \\
x_2 + x_3 & \geq & 1 \\
x_5, x_6 & \geq & 0 \\
x_1, x_2, x_3, x_4, x_5, x_6 & \leq & 1
\end{array}
$$

Abbildung 2.7:

[12] Vgl. [CR88a]
[13] Man sollte bedenken, daß eine Ungleichung, die facetteninduzierend für $\mathcal{P}^+_{\text{ST}}(D_G, T)$ ist, dies nicht notwendigerweise auch für $\mathcal{P}_{\text{ST}}(D_G, T)$ ist.

Die in die Wurzel zeigenden Bögen sind insofern herauszuheben, als sie in keiner optimalen Lösung enthalten sein können. Man sollte aufgrund des obigen Beispiels allerdings nicht annehmen, daß $\mathcal{P}_{\text{LP}}(D_G, T)$ immer gleich $\mathcal{P}_{\text{ST}}(D_G, T)$ ist.[14]

Wie Goemans und Myung in [GM93] gezeigt haben, ist $\min c^T x$, $x \in \mathcal{P}_{\text{LP}}(D_G, T)$ unabhängig von der Wahl der Wurzel r.

Nachfolgend wollen wir einige Zusammenhänge zwischen dem gerichteten und dem ungerichteten Modell mit Knotenvariablen aufzeigen.

Betrachtet man

$$y_k = \sum_{e \in \delta_k^-} x_e, \text{ für alle } k \in V \setminus \{r\} \tag{2.15}$$

wird klar, daß wir die Knotenvariablen des ungerichteten Modells durch die Summe der eingehenden Bögen im gerichteten Modell simulieren können. Dies ist eine Folge der Tatsache, daß es in einer optimalen Lösung S^* von $\text{BP}(\text{ST}(D_G, T, c))$ immer genau einen eingehenden Bogen für jeden Knoten aus $V(S^*)$ geben muß. Also sind

$$\sum_{a \in \delta_v^-} x_a \begin{cases} = 0 & \text{mit } v = r \\ = 1 & \text{für alle } v \in T \setminus \{r\} \\ \leq 1 & \text{für alle } v \in N \end{cases} \tag{2.16}$$

gültige (Un-)Gleichungen für jede minimale Lösung des gerichteten Steinerbaumproblems.

Da für alle Lösungen, die (2.16) erfüllen,

$$\sum_{a \in A} x_a = \sum_{k \in N} \sum_{a \in \delta_k^-} x_a + |T| - 1 \Rightarrow \sum_{k \in T} \sum_{a \in \delta_k^-} x_a = |T| - 1$$

gilt, wird offenbar Ungleichung (2.4) unter Verwendung von (2.15) auch von allen minimalen Lösungen des gerichteten Modells erfüllt.

Da jeder Steinerknoten, der Teil einer optimalen Lösung sein soll, mit mindestens zwei Knoten verbunden sein muß, und er maximal zu einem eingehenden Bogen inzident sein kann, muß es also mindestens einen ausgehenden Bogen geben.

Eine gerichtete Formulierung von Ungleichung (2.10) ist

$$\sum_{a \in \delta_v^+} x_a - \sum_{a \in \delta_v^-} x_a \geq 0, \text{ für alle } v \in N \tag{2.17}$$

und die Entsprechung von (2.9) ist dann

$$\text{für alle } i \in \delta_v^+ : \sum_{a \in \delta_v^-} x_a - x_i \geq 0, \text{ für alle } v \in N, \tag{2.18}$$

und die Ungleichung wird von einer minimalen Lösung mit Gleichheit für alle ausgehenden Bögen, die ungleich Null sind, erfüllt. In ähnlicher Weise kann man auch die Ungleichungen (2.5) und (2.6) für das gerichtete Problem formulieren.

Wir bezeichnen die Ungleichungen (2.16) bis (2.18) im weiteren als „Flußungleichungen“, weil sie auf der Überlegung beruhen, daß, wenn die Werte x_a, $a \in E$, einer zulässigen Lösung der Relaxierung als Fluß in D_G aufgefaßt werden, ein Fluß von genau[15] eins von der Wurzel zu jedem Terminal vorhanden sein muß.

[14] Bezüglich weiterer Klassen von facetteninduzierenden Ungleichungen für $\mathcal{P}^+_{\text{ST}}(D_G, T)$ siehe [CR94b].

[15] Das ist äquivalent damit, daß eine zulässige Lösung eine Arboreszenz induziert.

Vergleich der Modelle

Bevor wir uns für eines der vorgestellten Modelle entscheiden, sind zwei Fragen zu klären:

1. Welche Relaxierung ist die „Beste“, bzw. was ist

$$\text{argmax}_{P \in Q}\{\min\{c^T x \mid x \in P\}\} \text{ mit } Q = \{\mathcal{P}_{\text{LP}}(G,T), \mathcal{P}'_{\text{LP}}(G,T), \mathcal{P}_{\text{LP}}(D_G,T)\} \text{ ?}$$

2. Wie gut lassen sich (2.1), (2.3), (2.4), (2.5), (2.6), (2.10), (2.11), (2.16), (2.17) und (2.18) separieren ?

In [CR88a] zeigen Chopra und Rao, daß

$$\min\{c^T x \mid x \in \mathcal{P}_{\text{LP}}(G,T)\} \leq \min\{c^T x \mid x \in \mathcal{P}_{\text{LP}}(D_G,T)\}$$

gilt, selbst wenn man (2.2) und weitere bekannte Facetten zur Beschreibung von $\mathcal{P}_{\text{LP}}(G,T)$ hinzunimmt, und Goemans und Myung weisen in [GM93] nach, daß

$$\min\{c^T x \mid x \in \mathcal{P}'_{\text{LP}}(G,T)\} = \min\{c^T x \mid x \in \mathcal{P}_{\text{LP}}(D_G,T)\}$$

ist. Dabei ist aus den Ergebnissen des vorigen Abschnitts klar, daß sich die ungerichtete Formulierung in jedem Fall mindestens auf den Wert von $\mathcal{P}'_{\text{LP}}(G,T)$ heben läßt. Bleibt noch die Frage nach der Separierbarkeit der Ungleichungen.

Da uns die gerichtete Formulierung am einfachsten erschien und das durch (2.11) gegebene Separierungsproblem polynomial in $O(|V|^3)$ Schritten zu lösen ist und auch die Separierung der Flußungleichungen keine Probleme bereitet, haben wir uns für eine Verwendung dieses Modells entschieden und so in $\mathcal{P}_{\text{LP}}(D_G,T) \supseteq \mathcal{P}_{\text{ST}}(D_G,T)$ eine Relaxierung unseres ursprünglichen Problems gefunden, die wir bei der Implementierung unseres Schnittebenenverfahrens zugrundegelegt haben.

Kapitel 3

Reduktionsverfahren

In diesem Kapitel sollen Methoden untersucht werden, die mit polynomialem Aufwand die Anzahl der zur Lösung eines Steinerbaumproblems zu untersuchenden Knoten und Kanten verringern.

3.1 Theorie

Definition 3.1
Eine (zulässige) *Reduktion* ist eine Transformation eines Steinerbaumproblems $\mathrm{ST}(G,T,c)$ in ein Paar $(\mathrm{ST}(G',T',c'),c_r)$ mit $G'=(V',E')$ und $c_r \in \mathbb{R}_+$, $|V'| \leq |V|$, $|E'| \leq |E|$ und $|T'| \leq |T|$, so daß für mindestens eine optimale Lösung S^* von $\mathrm{ST}(G,T,c)$ und S'^* von $\mathrm{ST}(G',T',c')$ gilt: $c(S^*) = c'(S'^*) + c_r$.

Lemma 3.2
Sei $(\mathrm{ST}(G',T',c'),c_1)$ eine Reduktion von $\mathrm{ST}(G,T,c)$ und $(\mathrm{ST}(G'',T'',c''),c_2)$ eine Reduktion von $(\mathrm{ST}(G',T',c'),c_1)$, dann ist $(\mathrm{ST}(G'',T'',c''),c_1+c_2)$ eine Reduktion von $\mathrm{ST}(G,T,c)$.

Wir werden uns Bedingungen überlegen, die die folgenden Implikationen haben:

(i) Es gibt eine Kante $e \in E$ und mindestens eine optimale Lösung S^* von $\mathrm{ST}(G,T,c)$, in der die Kante e nicht enthalten ist. $(ST((V,E\setminus\{e\}),T,c),0)$ ist dann eine Reduktion.

(ii) Es gibt einen Knoten $v \in N$ und mindestens eine optimale Lösung S^* von $\mathrm{ST}(G,T,c)$, in der der Knoten v nicht enthalten ist. $(ST((V\setminus\{v\},E\setminus\delta_v),T,c),0)$ ist dann eine Reduktion.

(iii) Es gibt eine Kante $e=[u,v] \in E$ mit $u,v \in V$ und mindestens eine optimale Lösung S^* von $\mathrm{ST}(G,T,c)$, in der die Kante e enthalten ist. Sei dann $G' := G(\{u,v\}) = (W,F)$ der Graph, der durch Kontraktion von u und v entlang e entsteht. Es sei w der kontrahierte Knoten, dann ist $(ST(G',T',c),c_e)$ eine Reduktion mit $T' := T \cup \{w\} \setminus \{u,v\}$, falls $\{u,v\} \cap T \neq \emptyset$ und $T' := T$ sonst.

(iv) Es gibt zwei Kanten $e_1=[u,v]$ und $e_2=[v,w]$ mit $\{e_1,e_2\}=\delta_v$ und $v \in N$, $u,w \in V$ und mindestens eine optimale Lösung S^* von $\mathrm{ST}(G,T,c)$, für die gilt: $e_1 \in S^* \Leftrightarrow e_2 \in S^*$. Dann ist $(ST((V\setminus\{v\},E\cup\{[u,w]\}\setminus\delta_v),T,c'),0)$ eine Reduktion mit $c'_{[u,w]} := c_{e_1}+c_{e_2}$.

Um eine Vorstellung davon zu bekommen, was diese Reduktionen leisten können, betrachte man die Abbildungen[1] 3.1 und 3.2.

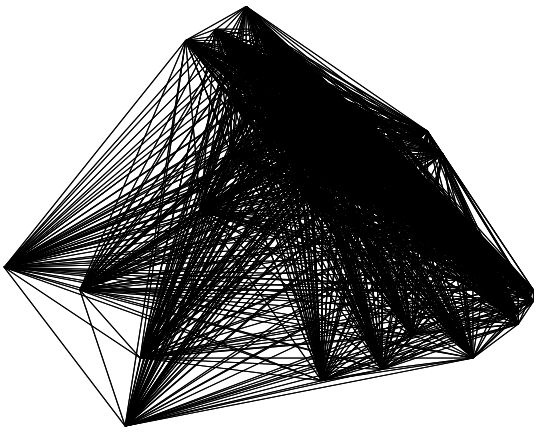


Abbildung 3.1: „Vorher“

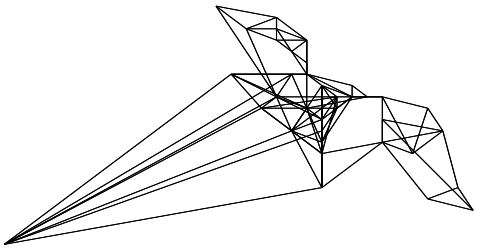


Abbildung 3.2: „Nachher“

Definition 3.3
Für ein Steinerbaumproblem $\mathrm{ST}(G,T,c)$ wollen wir die *Terminaldistanz* $s(u,v)$ zweier Knoten $u,v \in V$ mit $u \neq v$ definieren als die „kleinste maximale Entfernung“ zwischen zwei Terminalen auf einem Weg von u nach v:
Sei $H_{[u,v]} = (W,F)$ der durch einen $[u,v]$-Weg induzierte Subgraph von G und $T_W = W \cap T \cup \{u,v\}$. Dann sei

$$b(H_{[u,v]}) = \max \left\{ \sum_{e \in \bar{F}} c_e \;\middle|\; \bar{H} = (\bar{W},\bar{F}) \subseteq H_{[u,v]}, \bar{W} \subseteq W, \bar{F} \subseteq F, \bar{H} \text{ zusammenhängend} \right.$$
$$\left. \text{und für alle } \bar{w} \in \bar{W} \text{ gilt bezüglich } \bar{H} : \left\{ \begin{array}{ll} \bar{w} \in T_W, & \text{falls } |\delta_{\bar{w}}| = 1 \\ \bar{w} \notin T_W, & \text{falls } |\delta_{\bar{w}}| = 2 \end{array} \right\} \right\}$$

und

$$s(u,v) = \min\{b(H_{[u,v]}) \mid \text{über alle } H_{[u,v]} \text{ in } G\}.$$

In der Abbildung 3.3 ist ein Beispiel für die Terminal-Distanz zwischen u und v gezeigt. $s(u,v)$ gibt den größten Abstand zwischen zwei Terminalen an, den man zurücklegen muß, wenn man sich von u nach v bewegt.[2]

Es gilt offenbar: $s(u,v) \leq d(u,v) \leq c_{[u,v]}$.

[1] Es handelt sich hier um das Problem *br* von Carlos Ferreira. (Vollständiger Graph mit 58 Knoten).

[2] Man könnte s es auch als „Tankstellen“-Distanz bezeichnen, da $s(u,v)$ festlegt, für wieviele Einheiten Weg der Tank eines Fahrzeugs mindestens Treibstoff fassen muß, damit man von u nach v gelangen kann, wenn die Terminale sowie u und v Tankstellen sind.

[3] Wir wollen als *Test* einen Algorithmus bezeichnen, der Knoten oder Kanten ermittelt, die den Voraussetzungen für eine Reduktion entsprechen.

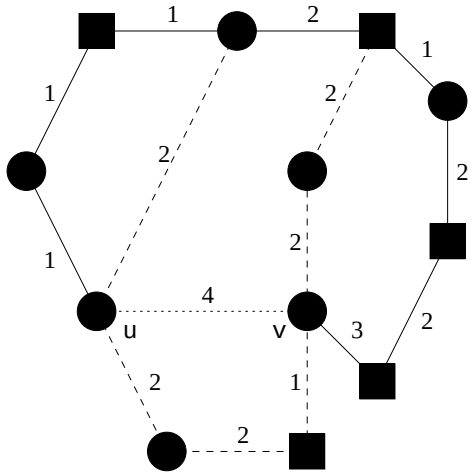


Abbildung 3.3: Graph mit $s(u,v) = 3$

In der Literatur wird eine Vielzahl von Reduktionstests[3] beschrieben. Wir haben, wenn möglich, die in den Artikeln genannten Namen mit aufgeführt, um das Durcheinander ein wenig zu lichten.

Wie sich schließlich herausgestellt hat, sind fast alle Reduktionen Spezialfälle von drei bzw. vier allgemeineren Aussagen (Sätze 3.4, 3.5, 3.6 und 3.10).

Satz 3.4 (Grad-Test I)
Gegeben sei ein Steinerbaumproblem $\mathrm{ST}(G,T,c)$, dann gilt für einen Knoten v mit

$|\delta_v| = 1,\ v \in N$ **:** v kann in keiner Optimallösung enthalten sein.

$|\delta_v| = 1,\ v \in T$ **:** Die Kante $e = [u,v] \in \delta_v$ muß in jeder zulässigen Lösung von $\mathrm{ST}(G,T,c)$ enthalten sein.

$|\delta_v| = 2,\ v \in N$ **:** Falls $v \in V(S^*)$ einer Optimallösung S^* von $\mathrm{ST}(G,T,c)$ ist, müssen beide Kanten $e_1 = [u,v]$, $e_2 = [v,w] \in \delta_v$ in S^* enthalten sein.

$|\delta_v| = 2,\ v \in T$**:** Seien $e_1 = [u,v]$, $e_2 = [v,w] \in \delta_v$ mit $c_{e_1} \leq c_{e_2}$. Falls $u \in T$, so gibt es eine Optimallösung[4] in der e_1 enthalten ist.

Dies ist der einfachste und schnellste der im folgenden vorgestellten Tests. Ein Durchlauf benötigt $O(|V|)$ Schritte, sofern die Gradzahlen der Knoten bekannt sind. Allerdings treten entsprechende Knoten in größerer Zahl nur bei dünnen, zufällig erzeugten Graphen oder als Folge anderer Reduktionen auf.

Die Verallgemeinerung für den Fall $v \in N$, $|\delta_v| \geq 3$ wird in [DV89b] als *Bottleneck Degree m Test* vorgestellt:

Satz 3.5 (Grad-Test II)
Gegeben ein Steinerbaumproblem $\mathrm{ST}(G,T,c)$ und ein Knoten $v \in N$ mit $|\delta_v| \geq 3$. Dann sei $G_v = (V_v, E_v)$ der vollständige Graph über der Knotenmenge $V_v = V(\delta_v)$, gewichtet mit den Terminaldistanzen $s_e = s(u,w)$ für alle $e = [u,w] \in E_v$ mit $u,w \in V_v$. Der Knoten v kann entfernt werden, wenn für jede Teilmenge $V'_v \subseteq V_v$ mit $|V'_v| \geq 3$ gilt:

$$\sum_{e \in E_v(V'_v)} s_e \leq \sum_{u \in V'_v} c_{[v,u]}$$

[4]Im Falle $c_{e_1} < c_{e_2}$ muß e_1 in jeder Optimallösung enthalten sein

mit $E'_v = \text{mst}((V'_v, E(V'_v)), s)$. Es müssen dann Kanten $[u, w]$ mit Kosten $c_{[u,w]} = c_{[u,v]} + c_{[v,w]}$ für alle $u, w \in V_v$ hinzugefügt[5] werden.

Beweis: Es genügt zu zeigen, daß der Grad, mit der v in der Lösung auftritt, null oder zwei ist. Ist letzteres der Fall, so gewährleistet das Einfügen der Kanten zwischen den Knoten in V_v die korrekte Lösung.
Es sei S^* eine optimale Lösung von $\text{ST}(G, T, c)$ und $v \in N$ ein Knoten für den die Bedingungen des Satzes zutreffen. Dann seien $\bar{S} = S^* \cap \delta_v$, $H_1 = (W_1, F_1), H_2 = (W_2, F_2)$ und $H_3 = (W_3, F_3)$ drei der $|\bar{S}|$ zusammenhängenden Komponenten von $H = (V \setminus \{v\}, S^* \setminus \bar{S})$. Desweiteren sei $w_i = W_i(\delta_v)$ und $e_i = [v, w_i]$ für $i \in \{1, 2, 3\}$. Da nach Voraussetzung $\min\{s(w_1, w_2) + s(w_1, w_3), s(w_1, w_2) + s(w_2, w_3), s(w_1, w_3) + s(w_2, w_3)\} \leq c_{e_1} + c_{e_2} + c_{e_3}$ gilt, ist es möglich, die drei Komponenten zu verbinden und dabei maximal zwei der drei Kanten e_1, e_2 und e_3 zu benutzen, ohne daß das Gewicht der Lösung größer wird. Sei o.B.d.A e_1 die unbenutzte Kante, dann kann eine andere Komponente aus H den Platz von W_1 einnehmen und der Vorgang solange wiederholt werden, bis gezeigt ist, daß maximal 2 Kanten gebraucht werden. ∎

Der Aufwand für diesen Test bei festgelegtem Grad $m = |\delta_v|$ ist polynomial (etwa $O(|V|^3)$ für $m = 3$), wächst aber exponentiell mit m.

Satz 3.6 (Terminal-Distanz-Test)
Die optimale Lösung S^* eines Steinerbaumproblems $\text{ST}(G, T, c)$ kann die Kante $[u, v] \in E$ nicht enthalten, wenn $s(u, v) < c_{[u,v]}$ ist.

Beweis: Es sei S^* eine optimale Lösung von $\text{ST}(G, T, c)$ mit $e = [u, v] \in S^*$ und $s(u, v) < c_e$. Dann gibt es einen von einem $[u, v]$-Weg induzierten Subgraphen $H_{[u,v]} = (W, F) \subseteq G$, für den $b(H_{[u,v]}) < c_e$ gilt. Da $T_W \subseteq V(S^*)$ für $T_W = W \cap T \cup \{u, v\}$ gilt, muß es einen $[s, t]$-Weg mit $s, t \in T_W$ der Länge $d(s, t) < c_e$ geben, der die beiden Komponenten von $V(S^* \setminus \{e\})$ verbindet. Sei $\bar{F}$ die Menge der Kanten auf diesem Weg, dann ist $S^* \setminus \{e\} \cup \bar{F}$ eine Lösung geringeren Gewichts. ∎

Mit einem zu diesem Zweck modifizierten Dijkstra-Algorithmus läßt sich der Test in $O(|V|^3)$ Schritten durchführen.

Der *Terminal-Distanz-Test* ist in [DV89a] als *Special Distance Test* beschrieben. Mehrere Spezialfälle werden in der Literatur beschrieben:

Folgerung 3.7 (Dreiecksungleichungs-Test)
Gegeben ein Steinerbaumproblem $\text{ST}(G, T, c)$ und eine Kante $e = [u, v] \in E$ mit $d(u, v) < c_e$. Dann kann e in der optimalen Lösung nicht enthalten sein.

Idee: Aus $d(u, v) < c_{[u,v]}$ folgt $s(u, v) < c_{[u,v]}$.

Dabei ist zu beachten, daß eine Verletzung der Dreiecksungleichung als Folge anderer Reduktionen auftreten kann (insbesondere bei Kontraktionen).

Dieser Test wird in [DV89b] als *Least Cost Test* und in [WS92] als *Longest Edge Reduction Type I* bezeichnet.

[5] Man beachte, daß insbesondere Kanten mit $s(u, w) < c_{[u,v]} + c_{[v,w]}$ wegen des Terminal-Distanz-Tests auch wieder entfernt werden können. Ebenso parallele Kanten.

Folgerung 3.8 (MST-Test)
Gegeben $\mathrm{ST}(G,T,c)$, sei $e=[u,v]\in E$, $H=T\cup\{u,v\}$ und $F=\mathrm{mst}((H,E(H)),c)\subseteq E$. Wir wollen annehmen, daß F eindeutig bestimmt sei.[6] Falls $e\notin F$, dann gibt es eine optimale Lösung S^*, in der e nicht enthalten ist.

Idee: Da e nicht Teil des minimal spannenden Baumes von H ist, muß es in dem durch $F\cup\{e\}$ induzierten Kreis eine darin enthaltene Kante f mit $c_f<c_e$ geben. Wegen $H=T\cup\{u,v\}$ folgt daraus aber $s(u,v)<c_e$.

In [BP87] wird dieser Test in drei Teile für $u,v\in T$ (*R-R Edge Deletion*), $u\in T$, $v\in N$ (*R-S Edge Deletion*) und $u,v\in N$ (*S-S Edge Deletion*) aufgeteilt. Die hier angegebene Form entspricht [DV89a].
Ein lokaler Sonderfall des *MST-Tests* wird in [WS92] als *Longest Edge Reduction Type II* und in [DV89b] als *Vertices nearer to K Test* beschrieben:

Folgerung 3.9 (Längstekanten-Test)
Gegeben $\mathrm{ST}(G,T,c)$. Es seien $u,v\in N$ und $t\in T$, dann existiert eine optimale Lösung S^* von $\mathrm{ST}(G,T,c)$ mit $e=[u,v]\notin S^*$, wenn $\max\{d(t,u),d(t,v)\}<c_e$ gilt.

Idee: Aus $\max\{d(t,u),d(t,v)\}<c_e$ mit $t\in T$ folgt sofort $s(u,v)<c_e$.

Wir haben zusätzlich eine vereinfachte Variante dieses Tests benutzt, bei der wir nur nach $\max\{c_{[t,u]},c_{[t,v]}\}<c_e$ suchen, was deutlich schneller ist, da keine Entfernungen zwischen den Knoten berechnet werden müssen.

Satz 3.10 (Terminalen-Entfernungs-Test)
Gegeben $\mathrm{ST}(G,T,c)$ und ein zusammenhängender Subgraph $H=(W,F)$ von G, für den $T\cap W\neq\emptyset$ und $T\setminus W\neq\emptyset$ gilt. Es seien $e=\mathrm{argmin}_{e\in\delta(W)}c_e$ und $f=\mathrm{argmin}_{f\in\delta(W)\setminus\{e\}}c_f$ eine kürzeste und zweitkürzeste[7] Kante aus dem durch W induzierten Schnitt.
Dann muß die Kante $e=[u,v]$ mit $u\in W$ und $v\in V\setminus W$ in einer optimalen Lösung S^* von $\mathrm{ST}(G,T,c)$ enthalten sein, falls gilt:

$$c_f\geq d_u+c_e+d_v$$

mit $d_u=\min\{d(t_1,u)\mid t_1\in T\cap W\}$ und $d_v=\min\{d(t_2,v)\mid t_2\in T\setminus W\}$.

Beweis: Es sei S^* eine optimale Lösung, die e nicht enthalte. Da mindestens eine Kante aus $\delta(W)$ benutzt werden muß, um die in W enthaltenen Terminale mit denen in $V\setminus W$ zu verbinden, ist $S^*\setminus\delta(W)\cup\{\text{Kanten des }[t_1,u]\text{-Weges}\}\cup\{e\}\cup\{\text{Kanten des }[v,t_2]\text{-Weges}\}$ eine mindestens so kurze Lösung von $\mathrm{ST}(G,T,c)$. ∎
Der Aufwand für diesen Test liegt bei $O(|V|^3)$. Er wird in [DV89b] als *Nearest Special Vertices Test* eingeführt.

Folgerung 3.11 (Terminalen-Aggregations-Test)
Gegeben $\mathrm{ST}(G,T,c)$ und $F=\mathrm{mst}(G,c)\subseteq E$. Dann existiert eine optimale Lösung S^* von $\mathrm{ST}(G,T,c)$, so daß jede Kante $e=[u,v]\in F$ mit $u,v\in T$ auch Element von S^* ist.

[6] Dies läßt sich, falls notwendig, durch eine totale Ordnung auf den Kanten erreichen.

[7] Hätten wir nicht vorausgesetzt, daß G zweifach zusammenhängend ist, könnten wir mit diesem Test auch *Steinerbrücken* entfernen, indem wir für den Fall $|\delta(W)|=1$, $c_f=\infty$ setzen.

Idee: Dieser Test entspricht dem Terminal-Entferungs-Test für den Fall $u, v \in T$, also $d_u = 0$, $d_v = 0$. Aus $e \in F$ folgt dann $c_e \leq c_f$ für ein $f \in E$ wie in Satz 3.10.

Der Aufwand für diesen Test ist im wesentlichen abhängig von der Berechnung des minimal aufspannenden Baumes und liegt daher bei $O(|V|^2)$. Er wird in [BP87] als *R-R Aggregation* beschrieben.

Folgerung 3.12 (Nähestes-Terminal-Test)
Gegeben $\mathrm{ST}(G, T, c)$. Sei $t \in T$ mit $|\delta_t| \geq 2$ und $u = \mathrm{argmin}_{u \in \delta_t} c_u$ und $v = \mathrm{argmin}_{v \in \delta_t \setminus \{u\}} c_v$ ein nächster und zweitnächster Nachbar von t.
Sei $s = \mathrm{argmin}_{s \in T \setminus \{t\}} d(u, s)$ ein abgesehen von t zu u nächstgelegenes Terminal. Falls $c_{[t,u]} + d(s, u) \leq c_{[t,v]}$, ist die Kante $e = [t, u]$ Teil einer optimalen Lösung S^* von $\mathrm{ST}(G, T, c)$.

Idee: Lokaler Sonderfall des Terminalen-Entfernungs-Tests.

Wir benutzen in der Implementierung ausschließlich eine vereinfachte Fassung, bei der nur direkte Verbindungen zwischen s und u überprüft werden. Dieser Test wird in [Bea84] und [DV89b] als *Nearest Vertex Test* und in [WS92] als *Closest Z-Vertices Reduction* beschrieben.

3.2 Praxis

Kosten-/Nutzenerwägungen

Bei jedem Test stellt sich die Frage, ob wir die Zeit, die verbraucht wird, im weiteren Verlauf des Verfahrens durch die eventuell verringerte Problemgröße wieder hereinholen.

Dazu muß überlegt werden, in welcher Reihenfolge und wie oft man die Reduktionsalgorithmen anwendet. Eine optimale Reihenfolge ist offenbar nicht im voraus zu ermitteln.
Aus unseren Erfahrungen geht hervor, daß es günstig ist, keine Reduktionstests mehr durchzuführen, wenn wahrscheinlich wird, daß nicht mehr „viel“ passiert.

Es sind daher zwei mögliche Abläufe implementiert worden: Ein „Schneller“ $\mathcal{S}$ und ein „Ausgiebiger“ $\mathcal{A}$. $\mathcal{S}$ wurde bei fast allen Lösungsläufen in Kapitel 7 verwendet, da er in kurzer Zeit einen großen Teil der möglichen Reduktionen findet. Bei $\mathcal{A}$ wird solange reduziert, wie noch eine Möglichkeit dazu besteht.

Auf Seite 25 sind die beiden Abläufe gezeigt. Der Grad-Test I wird immer solange durchgeführt, bis keine zutreffenden Knoten mehr gefunden werden.

Es ist noch anzumerken, daß unsere Implementierung des Terminal-Distanz-Tests mit zufällig perturbierten Kantengewichten arbeitet. Daher ist es nicht unsinnig, den Test mehrfach hintereinander auszuführen.

Ergebnisse

Alle in diesem Abschnitt gezeigten Resultate wurden mit dem Verfahren $\mathcal{A}$ erzielt. Es ging darum, herauszufinden, was an Reduktionen mit den vorgestellten Tests möglich ist.
Wir haben keine Laufzeiten angegeben, da der Algorithmus meist 99 % der Zeit – mitunter mehrere Stunden – aufgewendet hat, um die letzten 20 zu entfernenden Kanten zu finden.

Eine genauere Beschreibung der verwendeten Testdaten findet sich im Abschnitt 7.1 auf Seite 59ff.

In den nachfolgenden Tabellen bezieht sich die erste Hälfte der Spalten auf den ursprünglichen Graphen und die zweite auf das Ergebnis der Reduktionen.
Dabei ist $\delta_\downarrow$ der minimale und $\delta_\uparrow$ der maximale Grad aller Knoten im Graphen. $\delta_\emptyset$ bezeichnet den durchschnittlichen Grad.
Die Spalte % gibt an, wieviel Prozent der Kanten nach der Reduktion noch vorhanden sind.
Unter der Überschrift *Fixkosten* sind die Kosten verzeichnet, die zum Wert einer optimalen Lösung des reduzierten Steinerbaumproblems addiert werden müssen, um den Optimalwert des Ausgangsproblems zu erhalten. Ist der Eintrag fett dargestellt, so wurde das Problem vollständig reduziert und der angegebene Wert ist das Gewicht einer optimalen Lösung des Problems.
Erwartungsgemäß waren die Ergebnisse der Reduktionsverfahren sehr unterschiedlich. Bei sehr dünnen und bei vollständigen Graphen werden die besten Ergebnisse erzielt.
Wie man anhand der Tabellen 3.5, 3.6 und 3.7 sehr schön sehen kann, steigt der Reduktionserfolg mit der Anzahl der Terminalen. Das ist erklärlich, wenn man überlegt, daß durch das Hinzufügen eines Terminals zu einem Graphen die Terminal-Distanz der Knoten nur gleich bleiben oder kleiner werden kann. Ebenso wird die Anzahl der Terminal-Terminal Kanten nur größer oder bleibt gleich. Und diese Kanten können oft als zu einer optimalen Lösung gehörig identifiziert werden.

Bei vielen kleinen Datensätzen gelingt es, das Problem vollständig zu reduzieren und so zu lösen. Siehe die Tabellen 3.4 und 3.11.

Die Resultate bei vollständigen Graphen sind sehr gut, weil immer eine große Zahl von reduzierbaren Kanten vorhanden ist. Es erscheint unter diesem Gesichtspunkt wenig sinnvoll, Steinerbaumprobleme in vollständigen Graphen ohne Reduktionen lösen zu wollen.

Algorithmus: *Schnelle Reduktion* $\mathcal{S}$
Eingabe: Steinerbaumproblem $\mathrm{ST}(G, T)$.
Ausgabe: Ein um die durch das Verfahren identifizierten Knoten und Kanten reduziertes Steinerbaumproblem $\mathrm{ST}(G^*, T^*)$, sowie die für den fixierten Teil ermittelten Kosten.

Begin
Führe Grad-Test I aus.
Führe Terminalen-Aggregations-Test aus.
Führe Terminal-Distanz-Test aus.
Führe Grad-Test I aus.
Führe Nähestes-Terminal-Test aus.
Führe Terminal-Distanz-Test aus.
Führe Terminal-Distanz-Test aus.
Führe Nähestes-Terminal-Test aus.
Führe Grad-Test I aus.
End.

Algorithmus: *Ausgiebige Reduktion* $\mathcal{A}$
Eingabe: Steinerbaumproblem $\mathrm{ST}(G, T)$.
Ausgabe: Ein um die durch das Verfahren identifizierten Knoten und Kanten reduziertes Steinerbaumproblem $\mathrm{ST}(G^*, T^*)$ sowie die für den fixierten Teil ermittelten Kosten.

Begin
Führe Grad-Test I aus.
Do
Führe sechs mal den Terminal-Distanz-Test aus.
Führe Grad-Test I aus.
Führe Terminalen-Entfernungs-Test aus.
Führe Grad-Test II aus.
Führe Grad-Test I aus.
while einer der Tests erfolgreich war.
End.

Tabelle 3.1: Ablauf der Reduktionsverfahren

	$\mathrm{ST}(G,T,c)$ unreduziert						$\mathrm{ST}(G,T,c)$ reduziert							
Name	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	Fix-kosten
berlin	52	1326	16	51	51	51	47	138	14	3	12	5	10	118
br	58	1653	25	57	57	57	29	91	8	2	11	6	5	9892
gr	666	221445	174	665	665	665	536	2966	101	3	42	11	1	34878

Tabelle 3.2: Reduktionsergebnisse Testset X

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
mc2	120	7140	60	119	119	119	120	478	60	3	17	7	6	0
mc3	97	4656	45	96	96	96	97	1203	45	18	36	24	25	0
mc13	150	11175	80	149	149	149	149	623	80	2	16	8	5	1
mc11	400	760	213	2	4	3	40	58	26	2	6	2	7	10652
mc7	400	760	170	2	4	3	56	89	30	2	6	3	11	2472
mc8	400	760	188	2	4	3	58	88	35	2	5	3	11	1304

Tabelle 3.3: Reduktionsergebnisse Testset MC

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
b01	50	63	9	1	9	2	*	*	*				0	**82**
b02	50	63	13	1	8	2	*	*	*				0	**83**
b03	50	63	25	1	8	2	*	*	*				0	**138**
b04	50	100	9	1	10	4	*	*	*				0	**59**
b05	50	100	13	1	9	4	*	*	*				0	**61**
b06	50	100	25	1	9	4	16	25	9	2	5	3	25	67
b07	75	94	13	1	9	2	*	*	*				0	**111**
b08	75	94	19	1	7	2	*	*	*				0	**104**
b09	75	94	38	1	6	2	*	*	*				0	**220**
b10	75	150	13	1	10	4	28	53	8	2	7	3	35	35
b11	75	150	19	1	8	4	4	5	3	2	3	2	3	71
b12	75	150	38	1	8	4	*	*	*				0	**174**
b13	100	125	17	1	8	2	14	20	8	2	6	2	16	84
b14	100	125	25	1	7	2	20	28	10	2	4	2	22	155
b15	100	125	50	1	7	2	5	7	4	2	4	2	5	283
b16	100	200	17	1	9	4	44	90	9	2	8	4	45	52
b18	100	200	50	1	9	4	10	13	7	2	3	2	6	178

Tabelle 3.4: Reduktionsergebnisse Testset B

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
c01	500	625	5	1	12	2	113	202	5	2	8	3	32	25
c02	500	625	10	1	12	2	75	134	8	2	8	3	21	21
c03	500	625	83	1	10	2	36	51	23	2	6	2	8	554
c04	500	625	125	1	11	2	18	25	13	2	4	2	4	963
c05	500	625	250	1	10	2	*	*	*				0	**1579**
c06	500	1000	5	1	13	4	348	797	5	2	11	4	79	0
c07	500	1000	10	1	13	4	351	802	9	2	12	4	80	8
c08	500	1000	83	1	12	4	181	322	54	2	8	3	32	176
c09	500	1000	125	1	11	4	123	204	55	2	12	3	20	336
c10	500	1000	250	1	12	4	16	21	12	2	4	2	2	1038
c11	500	2500	5	2	22	9	495	1918	5	3	16	7	76	0
c12	500	2500	10	1	19	9	478	1623	10	3	15	6	64	0
c13	500	2500	83	1	21	9	277	555	61	2	9	4	22	65
c14	500	2500	125	2	20	9	30	42	20	2	4	2	1	280
c15	500	2500	250	2	20	9	*	*	*				0	**556**
c16	500	12500	5	2	71	49	500	2880	5	4	21	11	23	0
c17	500	12500	10	2	70	49	497	2353	10	3	18	9	18	0
c18	500	12500	83	2	74	49	428	1120	80	2	11	5	8	5
c19	500	12500	125	2	68	49	355	775	107	2	9	4	6	19
c20	500	12500	250	2	69	49	*	*	*				0	**267**

Tabelle 3.5: Reduktionsergebnisse Testset C

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
d01	1000	1250	5	1	13	2	228	430	5	2	11	3	34	0
d02	1000	1250	10	1	12	2	257	476	10	2	9	3	38	33
d03	1000	1250	167	1	12	2	*	*	*				0	**1565**
d04	1000	1250	250	1	10	2	8	11	5	2	5	2	1	1900
d05	1000	1250	500	1	10	2	8	12	6	2	6	3	0	3210
d06	1000	2000	5	1	12	4	742	1707	5	2	12	4	85	0
d07	1000	2000	10	1	12	4	717	1652	10	2	11	4	82	7
d08	1000	2000	167	1	16	4	202	328	87	2	11	3	16	501
d09	1000	2000	250	1	13	4	22	29	16	2	6	2	1	1360
d10	1000	2000	500	1	13	4	21	29	17	2	4	2	1	2033
d11	1000	5000	5	2	23	9	981	4133	5	3	19	8	82	0
d12	1000	5000	10	2	23	9	984	3518	10	3	15	7	70	0
d13	1000	5000	167	2	23	9	582	1153	141	2	8	3	23	76
d14	1000	5000	250	2	23	9	59	84	35	2	5	2	1	575
d15	1000	5000	500	2	21	9	10	14	8	2	5	2	1	1095
d16	1000	25000	5	2	73	49	1000	6946	5	5	25	13	27	0
d17	1000	25000	10	2	71	49	1000	6587	10	5	25	13	26	0
d18	1000	25000	167	2	77	49	879	2381	153	2	13	5	9	15
d19	1000	25000	250	2	72	49	843	2103	230	2	12	4	8	28
d20	1000	25000	500	2	74	49	*	*	*				0	**537**

Tabelle 3.6: Reduktionsergebnisse Testset D

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
e01	2500	3125	5	1	14	2	654	1246	5	2	13	3	39	9
e02	2500	3125	10	1	12	2	677	1263	9	2	11	3	40	50
e03	2500	3125	417	1	10	2	112	165	74	2	10	2	5	3380
e04	2500	3125	625	1	17	2	49	72	32	2	18	2	2	4831
e05	2500	3125	1250	1	11	2	7	10	5	2	6	2	1	8089
e06	2500	5000	5	1	17	4	1820	4274	5	2	16	4	85	0
e07	2500	5000	10	1	16	4	1863	4332	10	2	13	4	86	8
e08	2500	5000	417	1	16	4	642	1149	220	2	13	3	22	1233
e09	2500	5000	625	1	17	4	283	432	155	2	10	3	8	2698
e10	2500	5000	1250	1	16	4	14	20	11	2	7	2	1	5541
e11	2500	12500	5	2	24	10	2495	11541	5	3	20	9	92	0
e12	2500	12500	10	2	23	10	2483	10788	10	3	18	8	86	0
e13	2500	12500	417	2	26	10	1408	2742	353	2	13	3	21	211
e14	2500	12500	625	2	23	10	102	140	65	2	6	2	1	1565
e15	2500	12500	1250	2	23	10	*	*	*				0	**2784**
e16	2500	62500	5	2	75	49	2500	20694	5	5	30	16	33	0
e17	2500	62500	10	2	75	49	2500	17359	10	4	28	13	27	0
e18	2500	62500	417	30	73	50	2224	5994	394	2	14	5	9	35
e19	2500	62500	625	2	79	49	1608	3460	500	2	10	4	5	148
e20	2500	62500	1250	2	76	49	*	*	*				0	**1342**

Tabelle 3.7: Reduktionsergebnisse Testset E

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
p401	100	4950	5	99	99	99	58	124	5	3	7	4	2	4
p402	100	4950	5	99	99	99	56	113	5	3	7	4	2	3
p403	100	4950	5	99	99	99	65	149	5	3	8	4	3	0
p404	100	4950	10	99	99	99	*	*	*				0	**270**
p405	100	4950	10	99	99	99	*	*	*				0	**270**
p406	100	4950	10	99	99	99	55	111	8	2	6	4	2	39
p407	100	4950	20	99	99	99	51	92	14	2	6	3	1	108
p408	100	4950	20	99	99	99	*	*	*				0	**542**
p409	100	4950	50	99	99	99	7	10	5	2	6	2	1	800
p410	100	4950	50	99	99	99	*	*	*				0	**1010**
p455	100	4950	5	99	99	99	99	968	5	4	44	19	19	0
p456	100	4950	5	99	99	99	100	822	5	6	38	16	16	0
p457	100	4950	10	99	99	99	97	625	8	4	34	12	12	95
p458	100	4950	10	99	99	99	96	561	9	4	23	11	11	57
p459	100	4950	20	99	99	99	86	376	16	3	18	8	7	266
p463	200	19900	10	199	199	199	200	1963	10	6	44	19	9	0
p464	200	19900	20	199	199	199	190	1595	14	4	39	16	8	300
p465	200	19900	40	199	199	199	181	770	35	4	18	8	3	304
p466	200	19900	100	199	199	199	59	149	21	2	10	5	1	4337

Tabelle 3.8: Reduktionsergebnisse Testset P (Teil 1)

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
p601	100	180	5	2	4	3	34	63	5	2	5	3	35	2125
p602	100	180	5	2	4	3	33	58	5	2	5	3	32	7
p603	100	180	5	2	4	3	19	36	2	3	5	3	20	2414
p604	100	180	10	2	4	3	17	29	4	2	5	3	16	5647
p605	100	180	10	2	4	3	7	12	1	3	4	3	6	10355
p606	100	180	10	2	4	3	15	27	3	2	4	3	15	7794
p607	100	180	20	2	4	3	*	*	*				0	**15358**
p608	100	180	20	2	4	3	*	*	*				0	**14439**
p609	100	180	20	2	4	3	22	35	8	2	4	3	19	9147
p610	100	180	50	2	4	3	9	16	4	3	4	3	8	26594
p611	100	180	50	2	4	3	*	*	*				0	**26903**
p612	100	180	50	2	4	3	6	8	4	2	3	2	4	27051
p613	200	370	10	2	4	3	65	120	7	2	5	3	32	3107
p614	200	370	20	2	4	3	71	119	14	2	5	3	32	7131
p615	200	370	40	2	4	3	41	68	17	2	10	3	18	22131
p616	200	370	100	2	4	3	7	9	5	2	3	2	2	58188
p619	100	180	5	2	4	3	*	*	*				0	**7485**
p620	100	180	5	2	4	3	9	12	4	2	4	2	6	3095
p621	100	180	5	2	4	3	*	*	*				0	**8688**
p622	100	180	10	2	4	3	23	36	6	2	4	3	20	5982
p623	100	180	10	2	4	3	14	21	6	2	4	3	11	10400
p624	100	180	20	2	4	3	*	*	*				0	**20246**
p625	100	180	20	2	4	3	30	49	10	2	4	3	27	8169
p626	100	180	20	2	4	3	*	*	*				0	**22346**
p627	100	180	50	2	4	3	14	22	6	2	5	3	12	34553
p628	100	180	50	2	4	3	*	*	*				0	**40008**
p629	100	180	50	2	4	3	*	*	*				0	**43287**
p630	200	370	10	2	4	3	31	51	5	2	4	3	13	10782
p631	200	370	20	2	4	3	73	123	14	2	4	3	33	8719
p632	200	370	40	2	4	3	52	87	15	2	5	3	23	27371
p633	200	370	100	2	4	3	*	*	*				0	**86268**

Tabelle 3.9: Reduktionsergebnisse Testset P (Teil 2)

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
grd-16-9	144	263	7	2	4	3	90	159	6	2	4	3	60	3
grd-2-3	6	7	3	2	4	2	*	*	*				0	**3**
grd-3-4	12	17	4	2	4	2	*	*	*				0	**6**
grd-4-6	24	38	5	2	4	3	*	*	*				0	**9**
grd-6-9	54	93	7	2	4	3	18	27	4	2	4	3	29	7
grd-9-6	54	93	12	2	4	3	48	83	9	2	4	3	89	3

Tabelle 3.10: Reduktionsergebnisse Testset GRD

	ST(G,T,c) unreduziert						ST(G,T,c) reduziert							Fix-
Name	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	$\|V\|$	$\|E\|$	$\|T\|$	$\delta_\downarrow$	$\delta_\uparrow$	$\delta_\emptyset$	%	kosten
r01	15	22	5	2	4	2	*	*	*				0	**187**
r02	12	17	6	2	4	2	*	*	*				0	**164**
r03	28	45	7	2	4	3	*	*	*				0	**236**
r04	64	112	8	2	4	3	30	50	8	2	4	3	44	8
r05	12	17	6	2	4	2	*	*	*				0	**226**
r06	24	38	12	2	4	3	*	*	*				0	**242**
r07	30	49	12	2	4	3	*	*	*				0	**248**
r08	24	37	12	2	4	3	8	9	6	2	3	2	24	128
r09	15	22	7	2	4	2	*	*	*				0	**164**
r10	36	60	6	2	4	3	8	11	4	2	3	2	18	48
r11	30	49	6	2	4	3	*	*	*				0	**144**
r12	27	42	9	2	4	3	*	*	*				0	**180**
r13	42	71	9	2	4	3	24	37	8	2	4	3	52	30
r14	36	60	12	2	4	3	*	*	*				0	**260**
r15	100	180	14	2	4	3	34	54	11	2	4	3	30	44
r16	9	12	3	2	4	2	*	*	*				0	**160**
r17	48	82	10	2	4	3	7	11	3	3	4	3	13	127
r18	182	337	62	2	4	3	119	203	42	2	4	3	60	138
r19	168	310	14	2	4	3	36	59	8	2	4	3	19	97
r20	6	7	3	2	3	2	*	*	*				0	**112**
r21	15	22	5	2	4	2	*	*	*				0	**192**
r22	16	24	4	2	4	3	*	*	*				0	**63**
r23	16	24	4	2	4	3	*	*	*				0	**65**
r24	16	24	4	2	4	3	*	*	*				0	**30**
r25	9	12	3	2	4	2	*	*	*				0	**23**
r26	9	12	3	2	4	2	*	*	*				0	**15**
r27	16	24	4	2	4	3	*	*	*				0	**133**
r28	12	17	4	2	4	2	*	*	*				0	**24**
r29	9	12	3	2	4	2	*	*	*				0	**200**
r30	28	45	12	2	4	3	*	*	*				0	**110**
r31	130	237	14	2	4	3	65	112	14	2	4	3	47	28
r32	210	391	19	2	4	3	123	221	19	2	4	3	56	28
r33	132	241	18	2	4	3	75	135	11	2	4	3	56	113
r34	272	511	19	2	4	3	175	321	19	2	4	3	62	15
r35	240	449	18	2	4	3	123	218	18	2	4	3	48	8
r36	6	7	4	2	3	2	*	*	*				0	**90**
r37	49	84	8	2	4	3	*	*	*				0	**90**
r38	100	180	14	2	4	3	61	105	12	2	4	3	58	30
r39	100	180	14	2	4	3	51	89	10	2	4	3	49	58
r40	64	112	10	2	4	3	37	61	10	2	4	3	54	8
r41	144	263	20	2	4	3	99	175	16	2	4	3	66	44
r42	81	144	15	2	4	3	22	32	10	2	4	2	22	53
r43	195	362	16	2	4	3	122	215	16	2	4	3	59	11
r44	196	364	17	2	4	3	137	245	17	2	4	3	67	1
r45	270	507	19	2	4	3	180	329	18	2	4	3	64	34
r46	16	24	16	2	4	3	*	*	*				0	**150**

Tabelle 3.11: Reduktionsergebnisse Testset R

Kapitel 4

Heuristiken

In diesem Kapitel wollen wir uns mit polynomialen Algorithmen zur Erzeugung einer zulässigen Lösung befassen.
Da alle bekannten Algorithmen, die eine optimale Lösung des Steinerbaumproblems liefern, exponentiellen Aufwand haben, besteht seit langem ein Interesse an „guten“ Heuristiken.
Einen Überblick und Vergleich gängiger Verfahren findet sich in [RC86] und [WS92].

Aufgrund des Zusammenhanges zwischen einem Steinerbaumproblem $\mathrm{ST}(G, T, c)$ und dem minimal spannenden Baum von G bietet sich folgendes Verfahren als erster Ansatz an:
Bilde $F = \mathrm{mst}(G, c)$ und lösche dann alle Kanten, die zu einem Knoten $v \in N$ inzident sind, dessen Grad im Baum eins ist.
Die so erzeugte Lösung ist zweifelsfrei zulässig, hat aber den Nachteil, daß sie beliebig schlecht sein kann, wie man an nebenstehender Abbildung sieht.

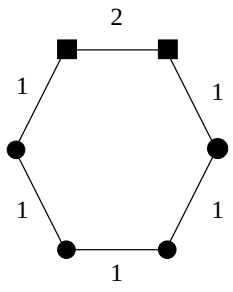


Abbildung 4.1:

Der Algorithmus kann aber auch zur Nachbearbeitung von heuristischen Lösungen eingesetzt werden. Hat eine Heuristik eine Menge $W \subseteq N$ von Knoten ausgewählt, so kann durch $S := \mathrm{mst}(T \cup W, E)$ eine Lösung berechnen werden, deren Gewicht, nach Löschung der angegebenen Kanten, in jedem Fall kleiner oder gleich dem der ursprünglich von der Heuristik ermittelten Lösung ist.

4.1 Weg-Entfernungs-Heuristiken

Mit den Weg-Entfernungs-Heuristiken liegt eine Klasse von Verfahren vor, die gute Ergebnisse (siehe Abschnitt 4.2) mit einem vertretbaren Aufwand (um $O(|V|^3)$) verbinden. Sie haben alle die in Tabelle 4.1 beschriebene Struktur. Man beginnt mit einem nur aus den Terminalen bestehenden Wald und verbindet dann sukzessive zwei Komponenten des Waldes, bis man einen zusammenhängenden Baum erhält.
Was die Heuristiken jetzt noch unterscheidet, ist die Auswahl im Schritt *Selektion eines Weges*. Wir wollen hier zwei der bekanntesten Ideen vorstellen:

Algorithmus: *Wege-Entfernungs-Heuristik*
Eingabe: Steinerbaumproblem $\mathrm{ST}(G, T)$.
Ausgabe: Eine zulässige Lösung S_H des Problems.

Begin *Wege-Entfernungs-Heuristik*
  Begin *Initialisierung*
    Beginne mit einem Graph $G_S = (T, \emptyset)$, einem Wald aus isolierten Terminalen.
  End.
  While G_S nicht verbunden ist **do**
    Begin *Selektion eines Weges*
      Füge zu G_S einen geeignet gewählten Weg in G zwischen
      zwei Komponenten von G_S hinzu.
    End.
  End.
  Begin *MST Berechnung*
    Füge zu G_S alle noch nicht enthaltenen Kanten aus G hinzu, die zwei Knoten
    in G_S verbinden und berechne einen minimal spannenden Baum von G_S.
    Lösche alle Kanten, die darin nicht enthalten sind.
  End.
  Begin *Beschneiden*
    Entferne sukzessive alle Knoten und die zugehörige Kante
    aus G_S, deren Grad eins ist und die keine Terminale sind.
  End.
  Setze S_H gleich der Menge aller in G_S enthaltenen Kanten.
End.

Tabelle 4.1: Ablauf einer Wege-Entfernungs-Heuristik

Kürzeste-Wege-Heuristik nach Takahashi und Matsuyama

Takahashi und Matsuyama geben in [TM80] eine Regel für die Selektion eines Weges:

Zu Beginn des Algorithmus wird ein beliebiger (geeignet gewählter) Knoten in G_S ausgesucht und markiert.
Dann wird bei der Selektion immer ein Terminal, das dem den markierten Knoten enthaltenden zusammenhängenden Subgraphen von G_S am nächsten steht, auf dem kürzesten Wege damit verbunden.

Es bildet sich also ein zusammenhängender Subgraph um den ursprünglich markierten Knoten, dem immer ein ihm am nächsten stehendes Terminal und die zum Zusammenhang notwendigen Knoten und Kanten hinzugefügt werden.

Wie in [TM80] gezeigt wird, beträgt das Verhältnis zwischen der Länge der von der Heuristik gelieferten Lösung S_{Heu} und der Optimallösung S_{Opt} schlechtestenfalls

$$\frac{c(S_{\text{Heu}})}{c(S_{\text{Opt}})} \leq 2\left(1 - \frac{1}{|T|}\right).$$

Die von der Heuristik gelieferte Lösung ist also nie mehr als doppelt so lang wie das Optimum. In den von uns getesteten Beispielen betrug die Abweichung allerdings maximal 7 %.

Die Komplexität des Algorithmus wird mit $O(|T||V|^2)$ angegeben.

Die Qualität der Lösung hängt stark von der Wahl des Startknotens ab. Dieses Problem kann auf Kosten der Laufzeit behoben werden, indem man die Heuristik mehrmals mit verschiedenen Knoten als Startpunkt aufruft. Es ist dabei nicht notwendig, daß dieser ein Terminal ist, da überflüssige Steinerknoten durch die abschließende MST-Berechnung entfernt werden.

Durchschnittliche-Entfernungs-Heuristik nach Rayward-Smith und Clare

Hier wird ein Knoten $v^* \in V$ gewählt, der in einer bestimmten Weise als der „Zentralste“ für alle Teilbäume in G_S gelten kann. Dann werden von den beiden v^* am nächsten stehenden Teilbäumen, die kürzesten Wege zu v^* gewählt.

In [RC86] wird als Maß für die „Zentralität“ eines Knotens folgende Funktion vorgeschlagen: Für jeden Knoten v sortiere die zusammenhängenden Subgraphen $G'_1 \ldots G'_k$ von G_S in aufsteigender Reihenfolge nach ihrer Entfernung zu v. Falls v Teil eines Subgraphen ist, ist dieser der erste in der Liste. Dann sei

$$f(v) = \min_{2 \leq r \leq k} \left\{ \sum_{i=1}^{r} \frac{d(v, G'_i)}{r-1} \right\}.$$

Nun wähle $v^* = \text{argmin}_{v \in V} f(v)$. Es ist dabei nicht immer nötig, alle $k-1$ Ausdrücke zu berechnen, um $f(v)$ zu ermitteln. Ist $v \in T$, dann ist $f(v) = d(v, G'_2)$ und ist $v \in N$, dann liefert das kleinste r, so daß der $(r+1)$-ste Ausdruck größer ist als der r-te, den Wert für $f(v)$.
Die in [WI88] gezeigte *Worst-Case* Schranke liegt hier bei

$$\frac{c(S_{\text{Heu}})}{c(S_{\text{Opt}})} \leq 2\left(1 - \frac{1}{b}\right) \leq 2\left(1 - \frac{1}{|T|}\right),$$

wobei b die Anzahl der Blätter der optimalen Lösung ist.

Die Komplexität des Algorithmus wird in [Ray83] mit $O(|T||V|^2 + \gamma)$ angegeben. Dabei ist γ der Aufwand zur Berechnung der kürzesten Wege zwischen allen Knotenpaaren also etwa $O(|V|^3)$.

4.2 Ergebnisse

Die von uns erzielten Resultate entsprechen den Ergebnissen der Untersuchungen in [RC86] und [WS92]. Die Durchschnittliche-Entferungs-Heuristik (DEH) liefert meist bessere oder gleich gute Lösungen wie die Kürzeste-Wege-Heuristik (KWH). Startet man die KWH aber

für mehrere oder alle Knoten, so liefert sie in allen bis auf neun der von uns getesteten Fälle gleich gute oder bessere Ergebnisse als die DEH bei vergleichbaren Aufwand. Einschränkend muß gesagt werden, daß in [RC86] noch Hinweise für eine weitere Verbesserung der Heuristik gegeben werden, die nicht in unsere Implementierung eingeflossen sind.[1]
Die Tabellen zeigen die Ergebnisse der Heuristiken für die in Abschnitt 7.1 vorgestellten Probleminstanzen.
Die Spalten $|V|$, $|E|$ und $|T|$ zeigen die Größe des bearbeiteten Problems. Die Spalte H_{RC} zeigt den von der DEH ermittelten Wert, die Spalten H^1_{TM}, H^{10}_{TM} und $H^{\forall}_{\mathrm{TM}}$ geben den von der KWH ermittelten Wert für einen, zehn und $|V|$ Startknoten. Die Startknoten wurden zufällig ausgewählt, allerdings ist der beim Lauf mit einem Startknoten verwendete Knoten immer auch einer der für den Lauf mit zehn Knoten gewählten.

Die Spalten $\%_{\mathrm{RC}}$, $\%^1$ $\%^{10}$ und $\%^{\forall}$ geben die prozentuale Abweichug der jeweiligen Heuristik von dem in der Spalte *Opt.* gezeigten Optimalwert an. Ein '*' in einer Spalte bedeutet, daß der Optimalwert gefunden wurde. Steht 0.0 in einer Spalte, so ist der ermittelte Wert nicht optimal, aber die Differenz zum Optimum liegt unter einem Promille.

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
mc11	40	58	26	11723	11723	11718	**11718**	11689	0.3	0.3	0.2	0.2
mc13	149	623	80	**92**	95	95	95	92	*	3.2	3.2	3.2
mc2	120	478	60	75	76	76	**74**	71	5.3	6.6	6.6	4.1
mc3	97	1203	45	**47**	48	48	48	47	*	2.1	2.1	2.1
mc7	56	89	30	**3418**	3516	3447	3447	3417	0.0	2.8	0.9	0.9
mc8	58	88	35	1568	1568	1567	**1567**	1566	0.1	0.1	0.1	0.1

Tabelle 4.2: Ergebnisse der Heuristik für Testset MC

Wie man sieht, finden die Heuristiken bei Graphen mit wenigen Terminalen ($|T| \leq 10$) häufig das Optimum.

Während des B&C-Verfahrens (siehe Kapitel 5 und 6) rufen wir die KWH in regelmäßigen Abständen mit 10 Startknoten auf, wobei die Kantengewichte anhand der aktuellen LP-Lösung modifiziert werden, um sie für den Algorithmus attraktiver zu machen.

[1] Ein implementierungstechnisches Problem bei der DEH ist die große Zahl von Knotenentfernungen, die zur Berechnung von $f(v)$ benötigt werden. Dies legt es nahe, mit einer Matrix aller Entfernungen zu arbeiten, was aber bei 5000 Knoten leicht zu einem Speicherbedarf von 100 MB führen kann. Wir haben dies auf Kosten der Laufzeit umgangen. Es scheint aber sinnvoll, vor weiteren Experimenten eine bessere Implementierung zu erstellen.

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
berlin	47	138	14	1069	1069	1069	**1048**	1044	2.3	2.3	2.3	0.4
br	29	91	8	13681	13682	13681	**13666**	13655	0.2	0.2	0.2	0.1
gr	536	2966	101	127048	123139	123139	**123036**	122467	3.6	0.5	0.5	0.5

Tabelle 4.3: Ergebnisse der Heuristik für Testset X

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
grd-16-9	90	159	6	**32**	32	32	**32**	32	*	*	*	*
grd-6-9	18	27	4	**18**	18	18	**18**	18	*	*	*	*
grd-9-6	48	83	9	**22**	22	22	**22**	22	*	*	*	*

Tabelle 4.4: Ergebnisse der Heuristik für Testset GRD

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
b06	16	25	9	124	124	122	**122**	122	1.6	1.6	*	*
b10	28	53	8	90	90	86	**86**	86	4.4	4.4	*	*
b11	4	5	3	**88**	90	88	**88**	88	*	2.2	*	*
b13	14	20	8	167	170	165	**165**	165	1.2	2.9	*	*
b14	20	28	10	236	235	235	**235**	235	0.4	*	*	*
b15	5	7	4	**318**	318	318	**318**	318	*	*	*	*
b16	44	90	9	**127**	134	133	**127**	127	*	5.2	4.5	*
b18	10	13	7	220	219	218	**218**	218	0.9	0.5	*	*

Tabelle 4.5: Ergebnisse der Heuristik für Testset B

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
c01	113	202	5	**85**	87	85	**85**	85	*	2.3	*	*
c02	75	134	8	**144**	144	144	**144**	144	*	*	*	*
c03	36	51	23	760	755	755	**754**	754	0.8	0.1	0.1	*
c04	18	25	13	1080	1080	1079	**1079**	1079	0.1	0.1	*	*
c06	348	797	5	**55**	56	55	**55**	55	*	1.8	*	*
c07	351	802	9	**102**	102	102	**102**	102	*	*	*	*
c08	181	322	54	**511**	511	511	**511**	509	0.4	0.4	0.4	0.4
c09	123	204	55	**712**	716	716	713	707	0.7	1.3	1.3	0.8
c10	16	21	12	**1093**	1095	1093	**1093**	1093	*	0.2	*	*
c11	495	1918	5	**32**	32	32	**32**	32	*	*	*	*
c12	478	1623	10	48	48	47	**46**	46	4.2	4.2	2.1	*
c13	277	555	61	**263**	266	264	**263**	258	1.9	3.0	2.3	1.9
c14	30	42	20	**324**	325	324	**324**	323	0.3	0.6	0.3	0.3
c16	500	2880	5	12	13	11	**11**	11	8.3	15.4	*	*
c17	497	2353	10	19	19	19	**18**	18	5.3	5.3	5.3	*
c18	428	1120	80	121	124	122	**120**	113	6.6	8.9	7.4	5.8
c19	355	775	107	**152**	154	153	**152**	146	3.9	5.2	4.6	3.9

Tabelle 4.6: Ergebnisse der Heuristik für Testset C

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
d01	228	430	5	108	107	107	**106**	106	1.9	0.9	0.9	*
d02	257	476	10	223	223	220	**220**	220	1.3	1.3	*	*
d04	8	11	5	**1935**	1935	1935	**1935**	1935	*	*	*	*
d05	8	12	6	**3251**	3254	3251	**3251**	3250	0.0	0.1	0.0	0.0
d06	742	1707	5	**67**	71	67	**67**	67	*	5.6	*	*
d07	717	1652	10	**103**	103	103	**103**	103	*	*	*	*
d08	202	328	87	**1078**	1084	1084	1080	1072	0.6	1.1	1.1	0.7
d09	22	29	16	**1450**	1451	1450	**1450**	1448	0.1	0.2	0.1	0.1
d10	21	29	17	**2111**	2113	2112	2112	2110	0.0	0.1	0.1	0.1
d11	981	4133	5	31	31	31	**29**	29	6.5	6.5	6.5	*
d12	984	3518	10	**42**	42	42	**42**	42	*	*	*	*
d13	582	1153	141	511	512	511	**509**	500	2.2	2.3	2.2	1.8
d14	59	84	35	**667**	669	667	**667**	667	*	0.3	*	*
d15	10	14	8	**1117**	1118	1117	**1117**	1116	0.1	0.2	0.1	0.1
d16	1000	6946	5	**13**	14	13	**13**	13	*	7.1	*	*
d17	1000	6587	10	24	23	23	**23**	23	4.2	*	*	*
d18	879	2381	153	**237**	242	240	**237**	223	5.9	7.9	7.1	5.9
d19	843	2103	230	325	330	323	**323**	310	4.6	6.1	4.0	4.0

Tabelle 4.7: Ergebnisse der Heuristik für Testset D

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
e01	654	1246	5	**111**	118	111	**111**	111	*	5.9	*	*
e02	677	1263	9	225	216	214	**214**	214	4.9	0.9	*	*
e03	112	165	74	4035	4037	4032	**4032**	4013	0.5	0.6	0.5	0.5
e04	49	72	32	5108	5111	5105	**5105**	5101	0.1	0.2	0.1	0.1
e05	7	10	5	**8129**	8130	8129	**8129**	8128	0.0	0.0	0.0	0.0
e06	1820	4274	5	**73**	76	76	**73**	73	*	3.9	3.9	*
e07	1863	4332	10	**145**	163	155	**145**	145	*	11.0	6.5	*
e08	642	1149	220	2670	2670	2664	**2662**	2641	1.1	1.1	0.9	0.8
e09	283	432	155	**3627**	3633	3632	**3627**	3604	0.6	0.8	0.8	0.6
e10	14	20	11	5601	5601	5600	**5600**	5600	0.0	0.0	*	*
e11	2495	11541	5	**34**	37	34	**34**	34	*	8.1	*	*
e12	2483	10788	10	68	68	68	**67**	67	1.5	1.5	1.5	*
e13	1408	2742	353	**1302**	1317	1310	1307	1280	1.7	2.8	2.3	2.1
e14	102	140	65	1735	1735	1734	**1734**	1732	0.2	0.2	0.1	0.1
e16	2500	20694	5	**15**	15	15	**15**	15	*	*	*	*
e17	2500	17359	10	**25**	26	26	**25**	25	*	3.8	3.8	*
e18	2224	5994	394	**601**	613	611	605	564	6.2	8.0	7.7	6.8
e19	1608	3460	500	786	791	787	**784**	758	3.6	4.2	3.7	3.3

Tabelle 4.8: Ergebnisse der Heuristik für Testset E

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
r04	30	50	8	258	258	258	**254**	254	1.6	1.6	1.6	*
r08	8	9	6	**236**	236	236	**236**	236	*	*	*	*
r10	8	11	4	184	185	177	**177**	177	3.8	4.3	*	*
r13	24	37	8	**150**	150	150	**150**	150	*	*	*	*
r15	34	54	11	**148**	148	148	**148**	148	*	*	*	*
r17	7	11	3	**200**	200	200	**200**	200	*	*	*	*
r18	119	203	42	413	406	406	**406**	404	2.2	0.5	0.5	0.5
r19	36	59	8	189	188	188	**188**	188	0.5	*	*	*
r31	65	112	14	260	265	259	**259**	259	0.4	2.3	*	*
r32	123	221	19	**315**	315	315	**315**	313	0.6	0.6	0.6	0.6
r33	75	135	11	276	277	269	**269**	268	2.9	3.2	0.4	0.4
r34	175	321	19	**246**	255	251	247	241	2.0	5.5	4.0	2.4
r35	123	218	18	155	152	152	**151**	151	2.6	0.7	0.7	*
r38	61	105	12	168	166	166	**166**	166	1.2	*	*	*
r39	51	89	10	**166**	166	166	**166**	166	*	*	*	*
r40	37	61	10	**155**	163	155	**155**	155	*	4.9	*	*
r41	99	175	16	231	224	224	**224**	224	3.0	*	*	*
r42	22	32	10	**153**	153	153	**153**	153	*	*	*	*
r43	122	215	16	262	268	259	**259**	255	2.7	4.9	1.5	1.5
r44	137	245	17	258	264	257	**256**	252	2.3	4.5	1.9	1.6
r45	180	329	18	225	230	225	**220**	220	2.2	4.3	2.2	*

Tabelle 4.9: Ergebnisse der Heuristik für Testset R

Name	$\|V\|$	$\|E\|$	$\|T\|$	H_{RC}	H^1_{TM}	H^{10}_{TM}	$H^{\forall}_{\mathrm{TM}}$	Opt.	$\%_{\mathrm{RC}}$	$\%^1$	$\%^{10}$	$\%^{\forall}$
p401	58	124	5	158	170	157	**155**	155	1.9	8.8	1.3	*
p402	56	113	5	**116**	116	116	**116**	116	*	*	*	*
p403	65	149	5	**181**	184	181	**181**	179	1.1	2.7	1.1	1.1
p406	55	111	8	**290**	290	290	**290**	290	*	*	*	*
p407	51	92	14	617	601	590	**590**	590	4.4	1.8	*	*
p409	7	10	5	974	974	964	**964**	963	1.1	1.1	0.1	0.1
p455	99	968	5	1191	1166	1166	**1138**	1138	4.5	2.4	2.4	*
p456	100	822	5	**1228**	1229	1229	**1228**	1228	*	0.1	0.1	*
p457	97	625	8	**1612**	1642	1612	**1612**	1609	0.2	2.0	0.2	0.2
p458	96	561	9	**1868**	1868	1868	**1868**	1868	*	*	*	*
p459	86	376	16	2348	2348	2348	**2345**	2345	0.1	0.1	0.1	*
p463	200	1963	10	**1519**	1538	1538	**1519**	1668	-9.8	-8.5	-8.5	-9.8
p464	190	1595	14	2591	2574	2574	**2553**	2591	*	-0.7	-0.7	-1.5
p465	181	770	35	3880	3880	3880	**3862**	3836	1.1	1.1	1.1	0.7
p466	59	149	21	6253	6253	6253	**6239**	5924	5.3	5.3	5.3	5.0
p601	34	63	5	**10230**	10230	10230	**10230**	10230	*	*	*	*
p602	33	58	5	**8083**	8445	8083	**8083**	8083	*	4.3	*	*
p603	19	36	2	**5022**	5022	5022	**5022**	5022	*	*	*	*
p604	17	29	4	11950	11397	11397	**11397**	11397	4.6	*	*	*
p605	7	12	1	**10355**	10355	10355	**10355**	10355	*	*	*	*
p606	15	27	3	**13048**	13048	13048	**13048**	13048	*	*	*	*
p609	22	35	8	18570	18570	18570	**18383**	18263	1.7	1.7	1.7	0.7
p610	9	16	4	**30161**	30161	30161	**30161**	30161	*	*	*	*
p612	6	8	4	**30258**	30350	30258	**30258**	30258	*	0.3	*	*
p613	65	120	7	**18429**	18595	18595	18595	18429	*	0.9	0.9	0.9
p614	71	119	14	27912	27912	27478	**27478**	27276	2.3	2.3	0.7	0.7
p615	41	68	17	43406	42805	42684	**42552**	42474	2.1	0.8	0.5	0.2
p616	7	9	5	62465	62263	62263	**62263**	62263	0.3	*	*	*
p620	9	12	4	**8746**	8746	8746	**8746**	8746	*	*	*	*
p622	23	36	6	**16546**	16546	16546	**16546**	15972	3.5	3.5	3.5	3.5
p623	14	21	6	**19496**	19496	19496	**19496**	19496	*	*	*	*
p625	30	49	10	**23078**	23078	23078	**23078**	23078	*	*	*	*
p627	14	22	6	**40647**	40647	40647	**40647**	40647	*	*	*	*
p630	31	51	5	**26125**	26125	26125	**26125**	26125	*	*	*	*
p631	73	123	14	40141	40141	40141	**39705**	39067	2.7	2.7	2.7	1.6
p632	52	87	15	**56562**	57684	57016	**56562**	56217	0.6	2.5	1.4	0.6

Tabelle 4.10: Ergebnisse der Heuristik für Testset P

Kapitel 5

Ein Verfahren zur Lösung des Steinerbaumproblems

Als einfachstes Verfahren bietet sich die explizite Enumeration aller möglichen Lösungen an.

Benutzen wir das ungerichtete Modell aus Kapitel 2 und ordnen jeder Kante $e \in E$ des Graphen eine binäre Variable $x_e \in \{0, 1\}$ zu, die genau dann eins ist, wenn die Kante in der Lösung enthalten sein soll, dann erhalten wir zu dem in Bild 5.1 gezeigten Graphen vier Variablen, deren mögliche Werte wir leicht aufzählen können.

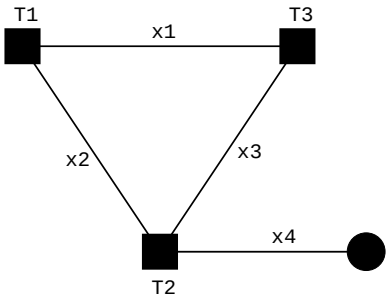


Abbildung 5.1:

x_1, x_2, x_3, x_4	$c^T x$	x_1, x_2, x_3, x_4	$c^T x$
0,0,0,0	Unzulässig	1,0,0,0	Unzulässig
0,0,0,1	Unzulässig	1,0,0,1	Unzulässig
0,0,1,0	Unzulässig	1,0,1,0	2
0,0,1,1	Unzulässig	1,0,1,1	3
0,1,0,0	Unzulässig	1,1,0,0	2
0,1,0,1	Unzulässig	1,1,0,1	3
0,1,1,0	2	1,1,1,0	Nicht minimal[1]
0,1,1,1	3	1,1,1,1	Nicht minimal[1]

Setzen wir das Gewicht aller Kanten auf eins, so kommen von 16 Lösungen nur sechs in Betracht und nur drei sind optimal. Darüber hinaus wird mit jeder zusätzlichen Kante die Anzahl der Lösungen, die aufzuzählen sind, verdoppelt. Das wirft die Frage auf, ob wir wirklich alle Werte enumerieren müssen oder ob wir nicht einen Teil ignorieren können.

5.1 Branch & Bound

Das Branch & Bound–Verfahren[2] versucht genau das zu erreichen.
Dazu wird das Problem in zwei „kleinere“ Teilprobleme zerlegt, z.B. durch Setzen einer Variablen auf eins in dem einen und auf null in dem anderen Teilproblem.[3]

[1] Kein Baum, da nicht kreisfrei. (Siehe Seite 8.)
[2] Wird auch als *Divide and Conquere*, *implizite Enumeration* usw. bezeichnet.
[3] Dies bezeichnet man als „Branching“.

Dieser Vorgang wiederholt sich dann mit den einzelnen Teilproblemen, bis sie so klein geworden sind, daß sie exakt gelöst werden können. Die vollständige Enumeration sieht dann wie in Abbildung 5.2 aus.

Wenn wir nun nach jedem Setzen einer Variablen überprüfen, ob es noch möglich ist, daß sich eine minimale Lösung ergibt, indem wir feststellen, ob die bisherige Teillösung kreisfrei ist oder ob Terminale abgeschnitten sind, können wir ganze Teilbäume ununtersucht lassen, wie in Bild 5.3 gezeigt wird.

Gleiches gilt, wenn wir feststellen, daß wir bereits eine zulässige Lösung erreicht haben. Die übrigen Variablen können dann auf null gesetzt und die Untersuchung des Zweiges beendet werden.

Haben wir einmal eine zulässige Lösung gefunden, besitzen wir eine obere Schranke[4] für den Optimalwert. Es ist nun unsinnig, noch Lösungen zu verfolgen, bei denen der Wert der festgelegten Kanten gleich oder größer als diese Schranke sind, weil wir in keinem Fall eine bessere zulässige Lösung in diesem Teilbereich finden können.[5]

Wir erhalten den in Bild 5.4 gezeigten Suchbaum.

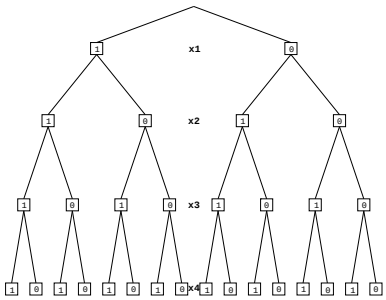


Abbildung 5.2:

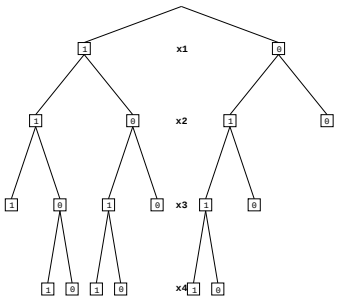


Abbildung 5.3:

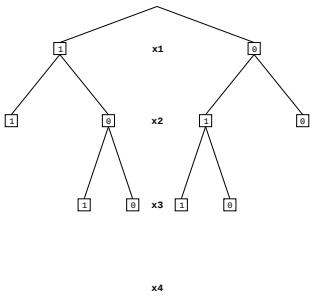


Abbildung 5.4:

Wie können wir diesen Algorithmus nun weiter verbessern ?

Einmal ist es unbefriedigend, auf die erste zulässige Lösung warten zu müssen, um eine obere Schranke zu erhalten. Besser wäre es, wir würden vor dem Beginn des Verfahrens versuchen, eine zulässige Lösung zu finden, um dann möglichst früh Teillösungen abschneiden zu können. Mögliche Wege, eine solche „Startlösung“ zu ermitteln, haben wir in Kapitel 4 gezeigt.

Zum anderen ist eine untere Schranke, die durch das Gewicht der festgelegten Variablen der Teillösung ermittelt wird zu niedrig. Offenbar muß noch etwas hinzukommen, denn sonst hätten wir schon eine zulässige Lösung.
Was gebraucht wird ist ein Algorithmus zur Ermittlung einer möglichst guten unteren Schranke.

[4]Wir gehen im folgenden davon aus, daß wir minimieren.

[5]Diese und die beiden vorangegangenen Überlegungen kann man dem Stichwort „Bounding“ zuordnen.

5.2 Schnittebenenverfahren

Das Schnittebenen- bzw. Cutting-Plane Verfahren ist eine Methode, eine untere Schranke für ein ganzzahliges Programm zu berechnen, indem man eine optimale Lösung für dessen LP-Relaxierung ermittelt.[6]

Wir hatten uns für das gerichtete Modell entschieden, und werden daher versuchen, den Optimalwert der LP-Relaxierung $\mathrm{LP}(\mathrm{ST}(D_G, T, c))$ von $\mathrm{BP}(\mathrm{ST}(D_G, T, c))$ zu berechnen.

Gegeben ein Steinerbaumproblem $\mathrm{ST}(D_G, T, c)$, dann könnte man alle gerichteten kantenminimalen Schnitte von einer beliebigen Wurzel $r \in V$ zu allen Terminalen $t \in T$ berechnen und das aus den durch die Schnitte induzierten Ungleichungen und c gebildete lineare Programm lösen.
Da die Anzahl der Schnitte aber exponentiell anwächst, verbietet sich diese Vorgehensweise für größere Probleme.

Stattdessen beginnen wir mit einem, abgesehen von den trivialen Ungleichungen $0 \leq x_a \leq 1$ für alle $a \in A$, leeren LP und fügen Ungleichungen nur dann hinzu, wenn sie von der aktuellen LP-Lösung verletzt werden. Das so entstandene LP wird wieder gelöst und der Vorgang solange iteriert, bis entweder

- der LP-Lösungsvektor $x \in \mathcal{P}_{\mathrm{ST}}(D_G, T)$ ist, also einer zulässigen Lösung entspricht. Diese ist dann optimal für $\mathrm{ST}(D_G, T, c)$. Oder

- für den Zielfunktionswert Z_{LP} der LP-Lösung $\lceil Z_{\mathrm{LP}} \rceil \geq Z^*$ gilt, wobei Z^* der Wert der besten bekannten Lösung von $\mathrm{ST}(D_G, T, c)$ ist. Es kann dann keine bessere Lösung für $\mathrm{ST}(D_G, T, c)$ mehr gefunden werden. Oder

- keine verletzten Ungleichungen mehr gefunden werden können. Sei x die optimale Lösung des linearen Programms, dann ist $\lceil c^T x \rceil$ eine untere Schranke für den Wert einer Lösung von $\mathrm{ST}(B(G), T, c)$.

Offenbar ist es von der Qualität der LP-Relaxierung, d.h. dem Abstand zwischen den Optima von $\mathcal{P}_{\mathrm{LP}}(D_G, T)$ und von $\mathcal{P}_{\mathrm{ST}}(D_G, T)$ bezüglich des Kostenvektors c abhängig, ob das Schnittebenenverfahren in der Lage ist, ein Problem zu lösen. Es liefert aber in jedem Fall eine untere Schranke für den Optimalwert von $\mathrm{ST}(D_G, T, c)$.

Aus einem gewissen Blickwinkel besteht kein Unterschied zwischen der iterativen Vorgehensweise und dem einmaligen Lösen des gesamten Systems. Benutzt man zur Lösung des linearen Programms den dualen Simplex Algorithmus, dann kann man das Hinzufügen der verletzten Ungleichungen auch als externes duales Pricing betrachten[7].

[6] Wir werden hier nur die Idee des für unsere Zwecke benutzten Schnittebenenverfahrens vorststellen. Für eine allgemeinere Sicht des Themas sei auf [PR91] und [GH88] verwiesen.

[7] Um so mehr, als man Ungleichungen, die von der LP-Lösung nicht mit Gleichheit erfüllt werden, auch aus dem System entfernen kann. Wir werden darauf in Kapitel 6 noch eingehen.

5.3 Branch & Cut

Kombinieren wir nun ein Branch & Bound- mit einem Schnittebenenverfahren, so erhalten wir ein Branch & Cut-Verfahren.

Eine Übersicht über den Ablauf eines typischen B&C-Programms bietet Abbildung 5.5. Dabei bezeichnet Z^* die beste bekannte obere Schranke für die Zielfunktion und Z_{LP} die durch das Lösen des LP erhaltene untere Schranke.

Man kann B&C als einen Spezialfall des B&B-Verfahrens sehen, der die untere Schranke mittels eines Schnittebenenverfahrens bestimmt.
Dabei ist zu berücksichtigen, daß der B&B-Teil oft überhaupt nicht zum Einsatz kommt, da je nach Problem und Güte des Schnittebenenverfahrens dieses in vielen Fällen schon eine Optimallösung liefert.

Wie wir noch in Kapitel 7 sehen werden, verbraucht ein B&C-Verfahren einen großen Teil seiner Laufzeit beim Lösen von linearen Programmen.

Mit den Details einer Implementierung, z.B. in welcher Reihenfolge die Teilprobleme abgearbeitet werden und wie man verletzte Ungleichungen findet, werden wir uns in Kapitel 6 beschäftigen.

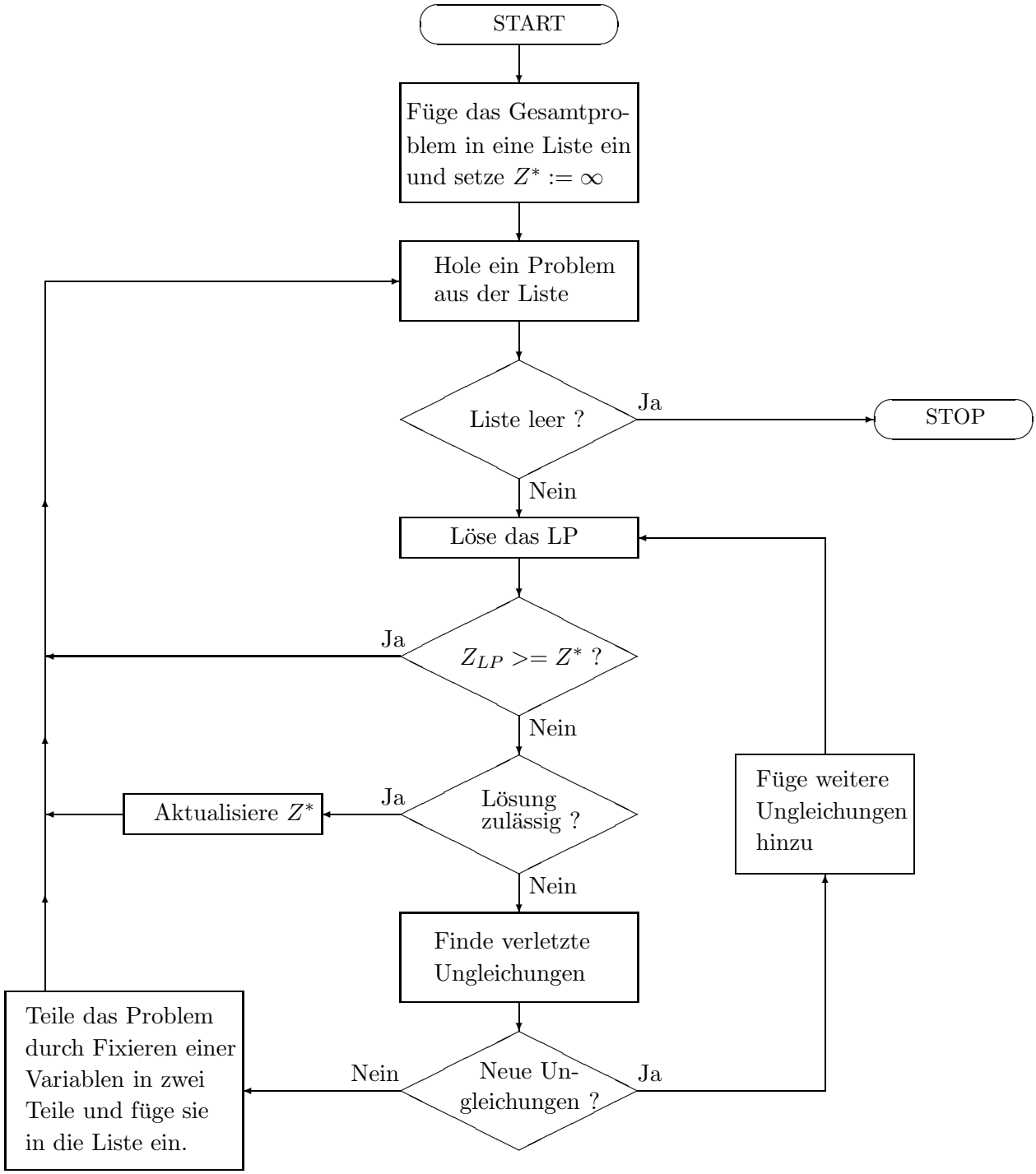


Abbildung 5.5: Flußdiagramm eines Branch & Cut Verfahrens

Kapitel 6

Die Implementierung

Nachdem wir in den vorhergehenden Kapiteln einen Überblick über den Aufbau eines B&C-Verfahrens und die zugrundeliegenden mathematischen Überlegungen gegeben haben, wollen wir jetzt Details einer Implementierung eines B&C-Verfahrens beschreiben, das versucht den Wert einer optimalen Lösung von Steinerbaumprobleminstanzen zu berechnen.

Die Implementierung eines solchen Algorithmus zeigt deutlich die Schwierigkeiten, die mit komplexen Verfahren verbunden sind.

Da das Gesamtverfahren aus mehreren theoretisch getrennten Teilen besteht, möchte man diese modular aufbauen und möglichst weit gegeneinander abschirmen.
Dem steht entgegen, daß jeder Teil des Verfahrens die anderen beeinflußt, so daß man zu dem Schluß gelangen kann, daß hier nicht mehrere Verfahren mit oder nacheinander arbeiten, sondern ein großes Ganzes wirkt, das nur so gut ist, wie seine schwächste Komponente.

Obgleich die im Konrad-Zuse-Zentrum für die Erstellung dieser Arbeit zur Verfügung stehenden Rechner kaum Wünsche offen ließen, ist es doch ein großer Unterschied, ob man Programme für Übungsaufgaben schreibt, oder ob die Datenmengen sich in Größenordnungen bewegen, bei denen man von den verfügbaren Ressourcen Speicherplatz und Rechenleistung nichts zu verschenken hat.

Eine andere Erkenntnis, die man bei größeren Projekten schnell erlangt, ist, daß es unmöglich ist, jedes Rad neu zu erfinden. Bei diesem Schnittebenenverfahren betrifft das den LP-Löser.

Hier stand mit R. E. Bixby's CPLEX ein Programm zur Verfügung, dessen Geschwindigkeit und Stabilität es erst möglich gemacht haben, Probleme dieser Größe mit Schnittebenenverfahren zu bearbeiten.

Ein Kriterium bei der Entwicklung des Programms war Portabilität. Der in „C“ geschriebene Code ist, eine entsprechende CPLEX Bibliothek vorausgesetzt, mindestens auf folgenden Systemen lauffähig: SunOS, Solaris, HP-UX, SCO UNIX V/386, Linux und MSDOS. Hierbei war der GNU C Compiler eine unschätzbare Hilfe.

6.1 Datenstrukturen

Da wir Lösungen für das Steinerbaumproblem *in Graphen* suchen, ist es notwendig eine geeignete Datenstruktur zu wählen, um Graphen effizient speichern und bearbeiten zu können.

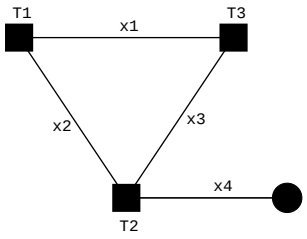
Wir haben für alle Algorithmen, die auf Graphen arbeiten, die nachfolgend beschriebene Datenstruktur verwendet:

Jede Kante wird zweifach gespeichert, je einmal mit vertauschten Anfangs- und Endknoten. So erhalten wir einen bidirektionalen Graphen. Mit jeder Kante sind außerdem Kosten und ggf. Kapazitäten gespeichert.

Für jeden Knoten halten wir dessen Grad, sowie, falls es sich um ein Terminal handelt, den Index der zugehörigen Terminalmenge fest[1]. Außerdem wird der Index der ersten eingehenden und der ersten ausgehenden Kante abgelegt.

In der Kantenliste finden sich dann neben den Anfangs- und Endknoten noch Einträge, die den Index der nächsten ein- bzw. ausgehenden Kante angeben. Ist einer dieser Einträge negativ, so handelt es sich bei der Kante um die letzte in der entsprechenden Liste für einen Knoten.

Betrachtet man den schon aus Kapitel 5 bekannten Graphen, so ergibt sich folgende Darstellung:



Knoten					
	1	2	3	4	
`term[]`	1	1	1	-1	Index der Terminalmenge
`grad[]`	2	3	2	1	Grad des Knotens
`inpbeg[]`	2	3	1	7	Index der ersten eingehenden Kante
`outbeg[]`	1	4	2	8	Index der ersten ausgehenden Kante

Kanten									
	1	2	3	4	5	6	7	8	
`tail[]`	1	3	1	2	2	3	2	4	Index des Fußknotens
`head[]`	3	1	2	1	3	2	4	2	Index des Kopfknotens
`ieat[]`	5	4	6	-1	-1	8	-1	-1	Nächste eingehende Kante
`oeat[]`	3	6	-1	5	7	-1	-1	-1	Nächste ausgehende Kante

Der Vorteil dieser Struktur liegt im direkten Zugriff auf die Mengen δ^+ und δ^- eines Knotens. Der Aufwand, eine weitere Kante zu einem Knoten hinzuzufügen, ist $O(1)$ und sie wieder zu löschen $O(\delta_v)$.

Insbesondere müssen die Kanten nicht in einer bestimmten Reihenfolge gespeichert werden. Werden Kanten gelöscht, so entstehen Lücken in der Datenstruktur. Hier ist es sinnvoll, die entsprechenden Kanteneinträge als „Unbenutzt“ zu markieren und mit diesen freien Einträgen eine Liste zu bilden, um ein schnelles Auffinden zu gewährleisten.

6.2 Der Branch & Bound-Algorithmus

Eine Übersicht über den Ablauf des hier implementierten B&B-Algorithmus findet sich in Tabelle 6.1 auf Seite 46.

[1] Beim Steinerbaumproblem gibt es immer nur eine Terminalmenge, aber bei einer Verallgemeinerung, dem Steinerbaumpackungsproblem in Graphen (siehe Anhang A.2 treten mehrere Terminalmengen auf. Hierbei ist es notwendig, daß die Terminalmengen disjunkt sind.

Algorithmus: *Branch & Bound*
Eingabe: Ein Steinerbaumproblem $\mathrm{ST}(D_G, T, c)$.
Ausgabe: Eine optimale Lösung S^* des Problems.

Sei Q eine Menge von Paaren (F, l), F eine Menge von Paaren (a, w) bestehend aus einem Bogen $a \in A$ und einem Fixierungswert $w \in \{0, 1\}$ von Variablenfixierungen und $l \in \mathbf{R}_+$ einer unteren Schranke für den Optimalwert von $\mathrm{LP}(\mathrm{ST}(D_G, T, c))$ bei Fixierung der Variablen $x_a = w$ für alle Paare $(a, w) \in F$.

Begin *Branch & Bound*
  Begin *Initialisierung*
    Setze $S := \emptyset$ (wobei $c(\emptyset) = \infty$).
    Setze $Q := \{(\emptyset, 0)\}$.
  End.
  While $Q \neq \emptyset$ **do**
    Begin *Evaluation*
      Entnehme ein geeignetes Element q aus Q und übergebe es an ein Schnittebenenverfahren zur Berechnung eines Lösungsvektors x von $\mathrm{LP}(\mathrm{ST}(D_G, T, c))$ unter Hinzunahme der durch q gegebenen Fixierungen.
    End.
    If $\mathrm{LP}(\mathrm{ST}(D_G, T, c))$ nicht unzulässig **then**
      If $c^T x < c(S)$ **then**
        If x zulässig und ganzzahlig **then**
          Begin *Bound*
            Setze S gleich der von x induzierten Lösung und
            lösche jedes Element aus Q mit $l \geq c(S)$.
          End.
        else
          Begin *Branch*
            Wähle einen Bogen $a \in A$ der nicht in q enthalten ist und
            setze $Q := Q \cup (F \cup (a, 0), c^T x) \cup (F \cup (a, 1), c^T x)$.
          End.
        End.
      End.
    End.
  End.
  Setze $S^* := S$.
End.

Tabelle 6.1: Ablauf des Branch & Bound–Verfahrens

Terminierung

Der vorgestellte Algorithmus terminiert, weil in jedem Schritt das Q entnommene Paar entweder verworfen wird oder durch zwei Paare mit jeweils einer weiteren fixierten Variablen ersetzt wird. Da insgesamt $|A|$ Variablen vorhanden sind, terminiert der Algorithmus nach maximal $2^{|A|}$ Iterationen.

Die beste Lösung ist dann optimal. Wurde keine zulässige Lösung gefunden, so besitzt das Problem keine.

Reihenfolge beim Verzweigen

Tabelle 6.1 läßt die Frage offen, wie ein *geeignetes* Element aus Q zu wählen wäre. Dabei geht es um die Festlegung der Reihenfolge in der die erzeugten Teilprobleme abgearbeitet werden.

Hierbei bietet sich aus Speicherplatzerwägungen die Tiefensuche an, daß heißt es wird immer das Paar $q = (F, l) := \text{argmax}_{q \in Q}|F|$ mit der größten Anzahl von Fixierungen gewählt. Es können dann zu einem beliebigen Zeitpunkt nie mehr als $2|A|$ Elemente in Q enthalten sein.

Andererseits gibt es keinen Grund anzunehmen, daß ein so gewähltes Element das für uns günstigste ist. Was wir suchen, ist ein Element, das eine Optimallösung des Gesamtproblems enthält. Da wir aber, wenn wir entscheiden könnten, in welchem Teil diese Lösung ist, das Problem schnell und einfach lösen könnten, hilft uns diese Überlegung nur wenig weiter.

Man kann nun heuristische Betrachtungen anstellen, um eine wahrscheinlich günstige Reihenfolge zu finden. Hier bietet es sich an, jeweils das Element $q \in Q$ mit der niedrigsten unteren Schranke l zu wählen. Falls es mehrere Kandidaten gibt, wählt man darunter das Element mit den meisten Fixierungen.

Dieses Vorgehen kann aber zu einer Breitensuche entarten. Es gilt dann $|F_1| - |F_2| \leq 1$ für alle $F_1 \in q_1$ und $F_2 \in q_2$ mit $q_1, q_2 \in Q$ und die Anzahl der Elemente in Q kann bis auf $2^{|A|}$ ansteigen.

Da der Schwerpunkt dieser Arbeit auf dem Schnittebenenverfahren liegt, sind keine ausgiebigen Tests mit verschiedenen Verzweigungsstrategien durchgeführt worden. Dazu kommt, daß nur sehr wenige der untersuchten Steinerbaumprobleminstanzen nicht vom Schnittebenenverfahren allein gelöst werden konnten.[2] Wir haben daher in allen angegebenen Beispielen eine Tiefensuche verwendet.

Das ist allerdings keine in der Implementierung festgelegte Forderung, hier kann durch Einsetzen einer Bewertungsfunktion eine beliebige Suchstrategie genutzt werden.

Alternativen beim Branching

Immer, wenn das Schnittebenenverfahren nur eine nichtganzzahlige Lösung zum Optimalwert liefert, „verkleinern“ wir das Problem Π durch Fixieren einer (weiteren) Variablen und

[2] Der Hauptgrund dafür, daß das Schnittebenenverfahren nicht ausreichte das Problem zu lösen, lag bei den Heuristiken, die nicht in der Lage waren, rechtzeitig eine obere Schranke zu liefern, die ein Verzweigen unnötig gemacht hätte. Daß die Tiefensuche das Auffinden von zulässigen Lösungen begünstigt, scheint daher eine vorteilhafte Eigenschaft zu sein.

untersuchen dann die beiden daraus entstehenden Probleme Π_1 und Π_2.

Das Fixieren einer Variablen x_a, $a \in A$, auf den Wert $k \in \{0, 1\}$ ist gleichbedeutend mit dem Einfügen einer Gleichung $x_a = k$ in das den Lösungsraum der LP-Relaxierung bezüglich Π_i, $i = 1, 2$ beschreibende Ungleichungssystem.

Eine weitere Möglichkeit wäre das Einfügen von Ungleichungen $\pi^T x \leq k$ in das Π_1 und $\pi^T x > k$ in das Π_2 beschreibende System mit $\pi \in \mathbb{R}^{|A|}$ und $k \in \mathbb{R}$.

Der Algorithmus und die Implementierung erlauben dieses *Verzweigen auf Ungleichungen*. Eine geeignete Wahl von π und k bleibt allerdings ungeklärt. Wir haben uns daher und wegen der nachfolgenden Überlegungen für das einfache Fixieren von Variablen entschieden.

Sei x eine nicht ganzzahlige optimale Lösung von $LP(\mathrm{ST}(D_G, T, c))$ und $F = \{e = [u, v] \in E \mid x_{(u,v)} + x_{(v,u)} \geq 0\}$, dann ist der Subgraph $H = (V(F), F) \subseteq G$ zusammenhängend aber nicht kreisfrei. Wäre er kreisfrei, müßten die Lösungen ganzzahlig sein, weil es sonst eine verletzte Steinerschnittungleichung gäbe.

Unser Ziel beim Branching sollte es also sein, diese Kreise zu unterbrechen. Fixieren wir eine Variable $a \in A$ mit $0 < x_a < 1$ auf null, so wird sicher ein Kreis in H unterbrochen. Bei einer Fixierung auf eins wird es unwahrscheinlicher, daß die nachfolgend ermittelten Optimallösungen den Kreis, zu dem a gehörte, beinhalten.

Eine gebrochene Lösung bedeutet im übertragenen Sinne, daß sich der LP-Löser wegen der vorliegenden Nebenbedingungen nicht für einen von mehreren möglichen Wegen entscheiden konnte. Durch die Fixierung wählen wir einen der Wege bzw. versperren ihn.

Wir haben deshalb in den vorliegenden Fällen nach dem „unentschiedensten“ Weg gesucht und immer die zum Bogen $a = \mathrm{argmin}_{a \in A} |x_a - 0.5|$ gehörige Variable fixiert.

6.3 Der Schnittebenenalgorithmus

Das Schnittebenenverfahren, das wir zur Ermittlung einer unteren Schranke für die im B&B-Algorithmus erzeugten Teilprobleme verwenden, arbeitet in etwa wie in Tabelle 6.2 gezeigt. Dabei ist anzumerken, daß in der augenblicklichen Implementierung noch keine Reduktionstests durchgeführt werden.[3]

Beim Aufruf der Heuristik werden die Bogengewichte nach der Formel $c_e^* = (1 - x_e)c_e$ modifiziert, so daß Bögen, die in der aktuellen Relaxierung als „wahrscheinlich“ gelten, bevorzugt benutzt werden.

Terminierung

Der in Tabelle 6.2 gezeigte Algorithmus terminiert, da dem LP nur Ungleichungen hinzugefügt werden, die die Koeffizienten 0, 1 und -1 enthalten und die von der aktuellen Lösung verletzt werden. Da keine Ungleichungen entfernt werden, kann es nach $3^{|A|}$ Iterationen keine Ungleichung mehr geben, die noch nicht enthalten ist, aber verletzt wird.

[3] Unter anderem wegen des seltenen Vorkommens von Fixierungen infolge reduzierter Kosten.

Algorithmus: *Schnittebenenverfahren*
Eingabe: Ein Steinerbaumproblem $\mathrm{ST}(D_G, T, c)$, eine Menge F von Fixierungen und eine obere Schranke Z^* für den Optimalwert von $\mathrm{ST}(D_G, T, c)$.
Ausgabe: Eine untere Schranke L für den Optimalwert von $\mathrm{ST}(D_G, T, c)$ bezüglich der durch F gegebenen Fixierungen.

Begin *Schnittebenenverfahren*
Beginne mit einem System (A, b), das die Ungleichungen $-x_a \leq 0$ und $x_a \leq 1$ sowie die aus den Fixierungen resultierenden Gleichungen enthält.
Do
Löse das durch $\min c^T x$, s.t. $Ax \leq b$ gegebene lineare Programm.
If LP hat keine zulässige Lösung **then**
Liefere als Ergebnis: „Es existiert keine zulässige Lösung“. **Beende.**
End.
If $\lceil c^T x \rceil \geq Z^*$ **then**
Setze $L := \lceil c^T x \rceil$. **Beende.**
End.
Begin *Heuristik*
Rufe die Primalheuristik unter Berücksichtigung der Lösung x auf.
If neue obere Schranke Z^* **then**
Begin *Reduced-Cost-Fixing*
Versuche Variablen aufgrund ihrer reduzierten Kosten zu fixieren.
If Variablen wurden fixiert **then**
Begin *Reduktionen*
Versuche mittels Reduktionstests weitere Variablen zu fixieren.
End.
End.
End.
End.
End.
Begin *Separierung*
Finde Ungleichungen, die von der aktuellen Lösung x verletzt werden und füge diese zu (A, b) hinzu.
End.
while eine Ungleichung wurde hinzugefügt.
Setze $L := \lceil c^T x \rceil$.
End.

Tabelle 6.2: Ablauf des Schnittebenenverfahrens

Tatsächlich tritt dieser Zustand natürlich viel früher ein, da wir nur bestimmte Klassen von Ungleichungen separieren und das Verfahren abbricht, falls keine zulässige Lösung des linearen Programms mehr gefunden werden kann.

Separierung

Die Separierungsverfahren sollen gültige Ungleichungen für das als binäres Programm formulierte Problem ermitteln, die bezüglich der aktuellen LP-Lösung verletzt sind.

In unserem Fall sind das vor allem die gerichteten Steinerschnitte. Das grundsätzliche Vorgehen ist sehr einfach: Wir ordnen die LP-Lösung als Kapazitäten u_a der Bogenmenge A von D_G zu und suchen dann jeweils zwischen der Wurzel r und einem Terminal t den kapazitätsminimalen Schnitt.

Die zugrundeliegende Überlegung besagt, daß wenn die Kapazität des minimalen r-t-Schnittes gleich eins ist, dann gibt es keinen Steinerschnitt, der t von r trennt.[4] Ist die Kapazität aber kleinergleich eins, haben wir sofort eine verletzte Ungleichung gefunden.

Die Berechnung des minimalen Schnittes ist in polynomialer Zeit möglich. Wir verwenden den in Tabelle 6.3 gezeigten und von Hao und Orlin in [HO92] vorgestellten *Preflow-Push-Algorithmus* (MinCut), der eine Laufzeitkomplexität von $O(|V|^3)$ hat. Der in 6.3 gezeigte Algorithmus entspricht dem in [HO92] vorgestellten[5].

Die Mengen $S(m) \subset V,\ m \leq l_{\max}$, die im Verlauf des Verfahrens erzeugt werden, stellen jeweils einen Schnitt im Graphen dar, über den kein weiterer Fluß mehr geschickt werden kann. x_a bezeichnet die Höhe des Flusses über einen Bogen $a \in A$.

Der Überschuß e_k eines Knotens k ist definiert als

$$e_k = \sum_{a \in \delta_k^-} x_a - \sum_{a \in \delta_k^+} x_a \text{ für alle } k \in V \setminus \{s\}$$

und der mögliche zusätzliche Fluß einen Bogens $a = (i, j)$ als

$$r_a = u_a - x_a + x_{(j,i)}.$$

Der Algorithmus erhält während seines Ablaufs drei Eigenschaften für jeden Bogen $a = (i, j)$ mit $a \in A$ und $i, j \in V$ aufrecht:

(i) $i, j \in W$ und $r_a > 0 \Rightarrow d_i \leq d_j + 1$.

(ii) $i \notin W$ und $j \in W \Rightarrow r_a = 0$.

(iii) $i \in S(m),\ j \in S(n)$ und $m < n \Rightarrow r_a = 0$.

[4] Die Kapazität des Schnittes kann, da wir minimieren, es sich um einen gerichteten Schnitt handelt und aufgrund der Art der im LP enthaltenen Ungleichungen, nicht größer als eins sein.

[5] Eine Darstellung des tatsächlich implementierten Algorithmus wäre mehr als doppelt so lang und würde wenig zum Verständnis beitragen.

Algorithmus: *Minimaler Schnitt (Preflow-Push)*
Eingabe: Ein bidirektionaler Graph $D_G = (V, A)$ mit Bogenkapazitäten $u_a \in \mathbb{N}_0$, $a \in A$ und zwei Knoten $s, t \in V$, zwischen denen der minimale Schnitt ermittelt werden soll.
Ausgabe: Eine Menge $W \subset V$ mit $t \in W$ und $s \notin W$, deren Grenze $\delta^-(W)$ zu $V \setminus W$ einen kapazitätsminimalen Schnitt zwischen s und t bildet.

Es seien d_k und e_k die Höhenmarkierung und der Überschuß eines Knotens $k \in V$, x_a und r_a der Fluß und der mögliche zusätzliche Fluß in Richtung des Bogens $a \in A$. Wir nennen einen Knoten $k \in V$ *aktiv*, wenn $k \in W \setminus \{t\}$ und $e_k > 0$ und einen Bogen $a = (i, j) \in A$ *zulässig*, wenn $i, j \in W$, $d_i = d_j + 1$ und $r_a > 0$.

Begin *Minimaler Schnitt*
  Begin *Initialisierung*
    Setze $x_a := 0$ und $r_a := u_a$ für alle $a \in A$.
    Schicke r_a Einheiten durch a von s weg, für alle Bögen $a \in \delta_s^+$.
    $l_{\max} := 0,\ S(0) := \{s\}$ und $W := V \setminus \{s\}$.
    $d_t := 0$ und $d_k := 1$ für alle $k \in V \setminus \{t\}$.
  end.
  While es gibt einen aktiven Knoten **do**
    Wähle einen aktiven Knoten k.
    If es gibt eine zulässigen Bogen $a = (k, i)$ **then**
      Schiebe $\Delta := \min\{e_k, r_a\}$ Einheiten entlang a von k nach i.
    else
      Begin *Markiere(k)*
        If $d_i \neq d_k$ für alle $i \in W \setminus \{k\}$ **then**
          $R := \{i \in W \mid d_i \geq d_k\}$.
          $l_{\max} := l_{\max} + 1,\ S(l_{\max}) := R$ und $W := W \setminus R$.
        else
          If es gibt keinen Bogen $a = (k, i)$ mit $r_a > 0$ und $i \in W$ **then**
            $l_{\max} := l_{\max} + 1,\ S(l_{\max}) := \{k\}$ und $W := W \setminus \{k\}$.
          else
            $d_k := \min\{d_i + 1 \mid a = (k, i) \in \delta_k^+,\ i \in W \text{ und } r_a > 0\}$.
          end.
        end.
      end.
  end.
end.

Tabelle 6.3: Preflow-Push-Algorithmus nach Hao und Orlin

Wir haben in unserer Implementierung einige Modifikationen vorgenommen, um die Geschwindigkeit zu erhöhen. Dazu werden zu Beginn die Markierungen $d_i,\ i \in V$, mittels einer Breitensuche mit den Entfernungen zur Senke t initialisiert und die Auswahl eines aktiven Knotens erfolgt durch geeignete Datenstrukturen in $O(1)$ Schritten[6].

Da wir den Algorithmus mindestens für jeden r-t Schnitt, $t \in T$, aufrufen, versuchen wir auszunutzen, daß in diesem Fall die Quelle s immer die Wurzel r ist. Wir verwenden das Ergebnis des vorherigen Laufes soweit wie möglich weiter, indem W durch Hinzunahme der Menge $S(l_{\max})$, $l_{\max} := l_{\max} - 1$, solange vergrößert wird, bis die neue Senke t' wieder in W enthalten ist. Dann wird nur W geeignet initialisiert und die Laufzeit für die Berechnung des Schnittes sinkt drastisch.
Zur Maximierung dieses Effekts, ist es sinnvoll, eine geeignete Reihenfolge für die Abarbeitung der Terminale zu finden. Ziel ist die Minimierung der notwendigen Hinzunahmen von Mengen zu W. Wir verwenden hierfür eine Greedyheuristik, die nach jedem Lauf des Algorithmus ein noch nicht untersuchtes Terminal ermittelt, das am „dichtesten“ an oder möglicherweise noch in W liegt.
Um nicht unnötig zu suchen, stellen wir zu Beginn der Separierung fest, ob es Wege mit Kapazität eins gibt und schließen Terminale, für die dies gilt, von der Untersuchung aus.

Nachdem wir nun eine Methode haben, verletzte Schnittungleichungen zu finden, gibt es noch einige Variationen, mit denen wir das Verfahren beeinflussen können:

Back-Cuts

Wenn wir nach einem r-t-Schnitt suchen, liegt es nahe, dies auch in umgekehrter Richtung zu tun, da die kapazitätsminimalen Schnitte in unserem Fall selten eindeutig[7] sind.

Wie in [CGR92] beschrieben wird, ist es dazu sinnvoll, bei der Zuordnung der LP-Lösung zu den Bögen die Richtungen zu vertauschen und als Kapazität dem Bogen (v, w) den Wert der LP-Lösung vom Bogen (w, v) zuzuordnen. Da unser Graph bidirektional ist, geht dies problemlos. Der Grund liegt darin, daß unser Verfahren gerichtete Wege *von* der Wurzel *zu* den Terminalen sucht und wir daher die Schnittungleichung ermitteln, indem wir alle Bögen aufsummieren, die bezüglich der beiden durch den Schnitt induzierten Subgraphen von dem die Wurzel enthaltenden Subgraphen in den sie nicht enthaltenden führen.

Nested-Cuts

Eine andere Möglichkeit ist das spekulative Erzeugen von Schnitten.

Wenn wir einen Schnitt gefunden haben und die zugehörige Ungleichung dem LP hinzufügen, setzt der LP-Löser oft nur eine der in den neuen Ungleichungen vorkommenden Variablen auf eins, und wir können beginnen, einen neuen Schnitt zu suchen.

[6] Wir wählen immer den Knoten mit der kleinsten Markierung aus. Das steht nicht im Einklang mit den theoretischen Erkenntnissen in [HO92] und [AMO92], die als günstigste Auswahl den Knoten mit der größten Markierung nennen. Für die von uns benutzten Daten haben Tests mit den vorliegenden Problemen deutlich die bessere Leistung unserer Auswahl gezeigt.

[7] Man stelle sich z.B. $u_a = 0$ für alle $a \in A$ vor.

Wir werden jetzt versuchen, dem vorzugreifen, indem wir alle zu einer gefundenen Ungleichung gehörigen Variablen auf eins setzen und dann nach weiteren suchen. Dabei ist die Menge der in den so gefundenen Ungleichungen vorkommenden Variablen disjunkt. Denn gäbe es eine gemeinsame Variable, könnte die neue Ungleichung nicht verletzt sein, weil alle Variablen der ursprünglichen Ungleichung auf eins gesetzt sind.

Dieses Verfahren läßt sich mit den oben angesprochenen *Back-Cuts* verbinden, so daß die Suche aus beiden Richtungen erfolgt. In Verbindung mit den Flußungleichungen können wir erreichen, daß die Wurzel bereits nach der ersten Iteration mit allen Terminalen verbunden ist. Wie man in Abbildung 6.5 in den Spalten „-BN“, „–N“, „-B-“ und „—“ sehen kann, nimmt die Anzahl der Iterationen nahezu um eine Größenordnung ab, wenn man eine der beiden Methoden benutzt und um eine weitere, wenn sie zusammen eingesetzt werden.
Wie zu erwarten, zeigt Bild 6.6 dann auch einen entsprechenden Anstieg in der Zeit je Iteration. Trotzdem führt insbesondere die Kombination beider Verfahren zu einer beträchtlichen Beschleunigung des B&C-Algorithmus.[8]

Creep-Flow

Nachdem wir zwei Methoden betrachtet haben, um die Anzahl der Schnitte zu erhöhen, versuchen wir nun, deren Qualität zu verbessern.

Wenn wir davon ausgehen, daß Ungleichungen mit weniger Koeffizienten günstig für den LP-Löser sind, dann müssen wir feststellen, daß die von uns gefundenen Schnitte zwar kapazitätsminimal aber nicht notwendigerweise bogenminimal sind.

Da die optimale Lösung eines bidirektionalen Steinerbaumproblems maximal $|V| - 1$ Bögen enthalten kann, der durchschnittliche Grad der Knoten in den untersuchten Problemen aber größer als drei ist, kann man, selbst wenn berücksichtigt wird, daß die LP-Lösung mehr nichtnullwertige Variablen enthalten kann, davon ausgehen, daß über 4/5 der Variablen einer LP-Lösung den Wert null haben und somit ohne Einfluß auf die Kapazität[9] des Schnittes sind.
Der MinCut-Algorithmus kann daher viele Bögen in den Schnitt aufnehmen, ohne daß sich dessen Kapazität ändert. Was wir jetzt erreichen möchten wäre ein bogenminimaler, kapazitätsminimaler Schnitt.

Dazu weisen wir jedem Bogen, dessen zugehöriger Wert in der LP-Lösung null ist, eine Kapazität von 10^{-6} zu. Das ist so wenig, daß es die Kapazität des Schnittes nicht merklich[10] beeinflußt, und zwingt andererseits den Algorithmus dazu, diesen „*Kriechfluß*“ zu berücksichtigen und die Anzahl der benutzten Bögen zu minimieren.

Wie man in Abbildung 6.4 sieht, reduziert sich die durchschnittliche Anzahl von Nichtnulleinträgen in den Ungleichungen drastisch, und Diagramm 6.3 zeigt, daß dies zu einer exponentiellen Verringerung der vom LP-Löser verbrauchten Zeit führt.

Der Nachteil dieser Idee ist die stark angestiegene Anzahl von Kanten, die der MinCut-Algorithmus berücksichtigen muß. Als Folge steigt die für die Separierung benötigte Zeit

[8] z.B. *gr* wurde ohne eines dieser Verfahren nach 5000 Iterationen erfolglos abgebrochen.
[9] Die Kapazität des Schnittes C bezüglich der LP-Lösung x ist $\sum_{a \in C} x_a$ wie vorher besprochen.
[10] Merklich würde bedeuten, es wird ein nicht kapazitätsminimaler Schnitt ermittelt.

deutlich an (siehe Bild 6.6).

Bewertung

Welche Methode oder Kombination von Methoden bringt die besten Ergebnisse ?

Dazu haben wir einige Untersuchungen angestellt, deren Ergebnisse in den Abbildungen auf den folgenden Seiten zu sehen sind. Es wurden dazu sechs Probleminstanzen ausgewählt und gelöst. Es ist zu beachten, daß, um die Ergebnisse sinnvoll darzustellen, in den Diagrammen 6.1, 6.2, 6.3 und 6.5 eine logarithmische Darstellungsweise gewählt wurde.

Leider zeigt sich, daß die Kombination des *Creep-Flow* Verfahrens mit *Back-* und *Nested-Cuts* nicht nutzbringend ist. Eine mögliche Erklärung, ist die Überlegung, daß die beiden letztgenannten Methoden darauf beruhen, daß der MinCut-Algorithmus nicht einen ganz bestimmten Schnitt findet, sondern ein (möglichst kleines) Gebiet um die Wurzel.

Die insgesamt besten Ergebnisse werden entsprechend Abbildung 6.1 nur mit der Anwendung des *Creep-Flow* Verfahrens erreicht. Diagramm 6.5 zeigt, daß, unabhängig von den anderen Verfahren, die Iterationszahlen immer dann am kleinsten waren, wenn solche kantenminimalen Schnitte verwendet wurden. In Verbindung mit der beschleunigten Lösung der LP's führt dies dann zu den besten Gesamtzeiten.

Wie Bild 6.2 zeigt, ist der Anteil der Zeit für die Separierung an der Gesamtzeit über die Verfahren hinweg relativ konstant, was aber sehr stark mit der unterschiedlichen Anzahl von Iterationen bei den einzelnen Verfahren zusammenhängt.

Es ist uns durch die beschriebenen Ideen gelungen, die Anzahl der für das Schnittebenenverfahren benötigten Iterationen um zwei bis drei Größenordnungen zu senken.

Das in den Diagrammen gezeigte Verhältnis von Separierungs- zu LP-Lösungszeit von etwa zehn zu eins war der Anlaß für die besprochenen Verbesserungen bei der Implementierung des MinCut-Algorithmus, so daß in der aktuellen Version ein weitaus besseres Verhältnis erreicht wird.

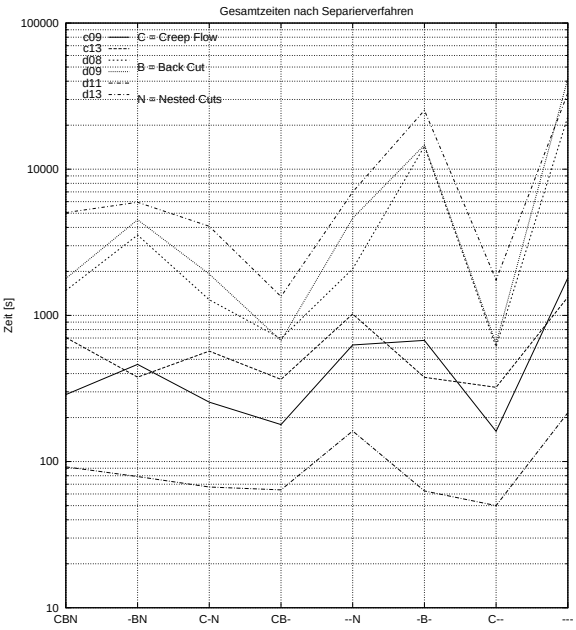


Abbildung 6.1:

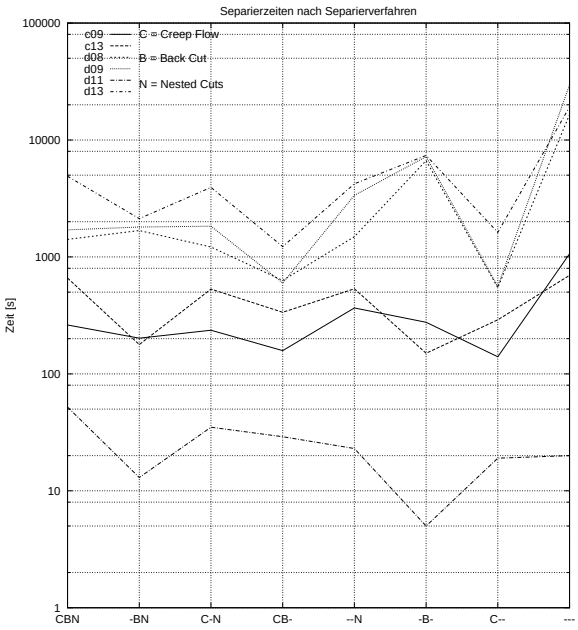


Abbildung 6.2:

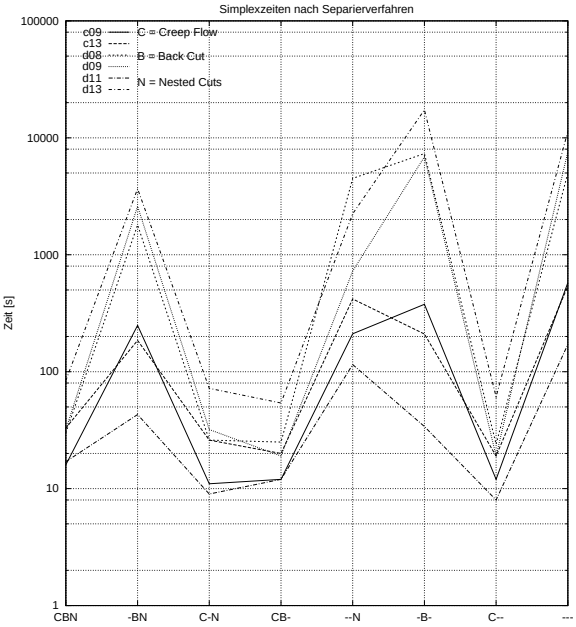


Abbildung 6.3:

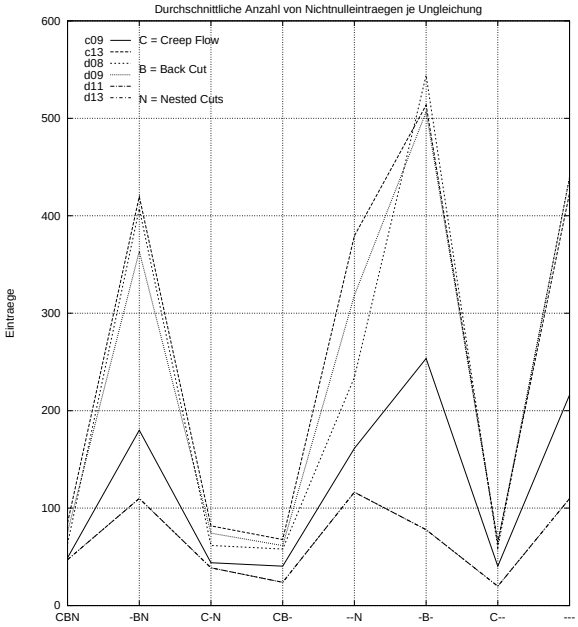


Abbildung 6.4:

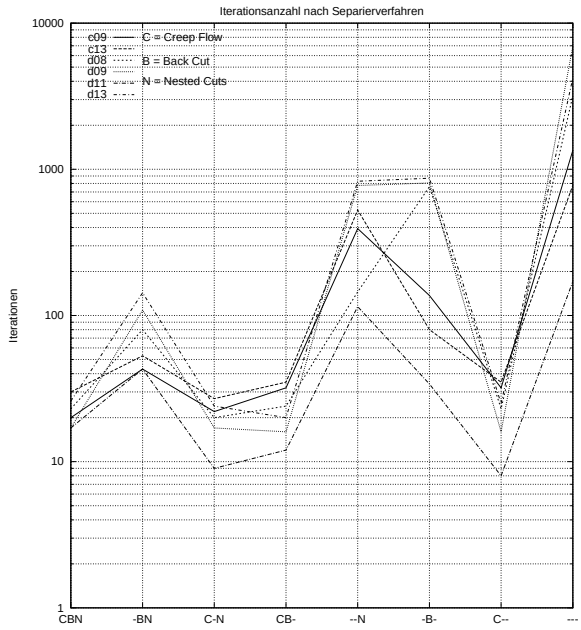


Abbildung 6.5:

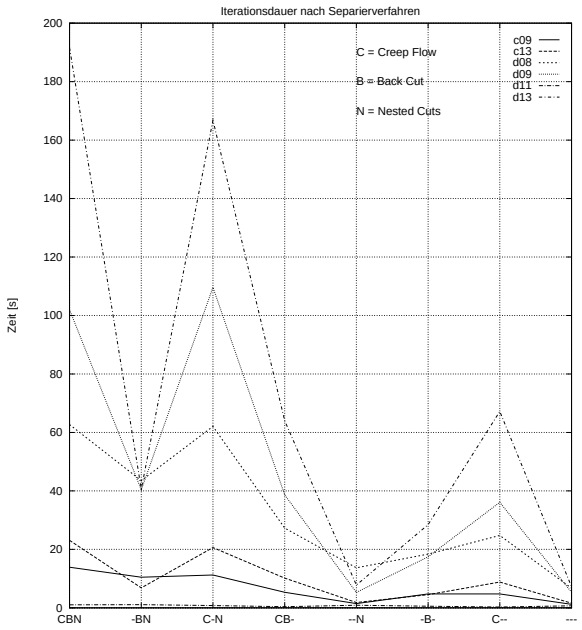


Abbildung 6.6:

Verweildauer von Ungleichungen im LP

Während des Schnittebenenverfahrens wird unser LP ständig größer, weil wir bei jeder Iteration Ungleichungen hinzufügen.

Dabei kommt es immer wieder vor, daß einige im LP enthaltene Ungleichungen von der aktuellen LP-Lösung nicht mehr mit Gleichheit erfüllt werden. Diese können wir dann entfernen, ohne daß der Zielfunktionswert sich ändert.

Einige Implementierungen sehen dabei einen als *Pool* bezeichneten Speicher vor, in dem Ungleichungen aufbewahrt werden, nachdem sie dem LP entnommen wurden, um diese, falls sie wieder verletzt werden, schnell separieren zu können.
Dies ist besonders dann sinnvoll, wenn der Aufwand, der zum Auffinden der Ungleichungen benötigt wurde, groß war oder wenn die Separierung nur heuristisch erfolgt und man nicht sicher sein kann, ob eine bestimmte Ungleichung nochmals separiert werden kann.

Da der Aufwand, eine verletzte Ungleichung zu finden, in unserem Fall verhältnismäßig gering ist, sind wir einen anderen Weg gegangen.
Wir entfernen Ungleichungen erst, wenn sie mehrfach hintereinander von der LP-Lösung nicht mit Gleichheit erfüllt wurden.

Perturbation

Die von unserem Schnittebenenverfahren erzeugten linearen Programme sind oft stark entartet (degeneriert), d.h. es existieren mehrere verschiedene Basen[11] zu einer Ecke des Polyeders.

[11] Bezüglich der Definition von Basen beim Simplexverfahren sei auf [Bix92] verwiesen.

So kann es passieren, daß das Simplexverfahren zwar die Basis ändert, die zugehörige Ecke aber gleich bleibt und keinen Fortschritt mehr erzielt.

Da wir das duale Simplexverfahren benutzen, können wir versuchen, durch Perturbation der Zielfunktion die Entartung des LP's zu reduzieren. Wir folgen dabei zuerst der in [Mar92] beschriebenen Methode, wobei der Zielfunktionsvektor $c \in \mathbb{R}_+^{|A|}$ durch

$$\tilde{c} = c - b * \varepsilon_a, \text{ mit } a \in A \text{ und } b = \min(10^{-5}, 2(|A| + 1) - 1)$$

ersetzt wird. ε_a ist eine gleichverteilte Zufallsvariable über dem Intervall $[0.0, 1.0)$.

Durch diese Wahl von $\tilde{c}$ wird gewährleistet, daß eine Optimallösung bezüglich $\tilde{c}$ auch optimal für den Zielfunktionsvektor c ist.

Trotz des Geschwindigkeitsgewinns, den wir mit der beschriebenen Methode beim Lösen der LP's erzielen, zeigen sich bei einigen, meist größeren Problemen, noch immer Entartungserscheinungen. Wir haben daher auch mit $b = 0.1$ experimentiert und gute Ergebnisse erhalten. Allerdings sind die Lösungen, die man erhält, möglicherweise nicht mehr optimal für das ursprüngliche Problem, und es ist nötig, die so erhaltenen Lösungen unter Verwendung eines zulässigen Zielfunktionsvektors zu reoptimieren.
Da unser Kostenvektor aber ganzzahlig ist, müßte eine „falsche" Lösung sich um mindestens zehn, bei Berücksichtigung von ε_a als gleichverteiltem Zufallswert, eher mehr als zwanzig Bögen von der optimalen Lösung unterscheiden.

Reduzierter Variablensatz

Eine andere Methode, die Größe des zu lösenden LP's klein zu halten, ist das Arbeiten auf einem reduzierten Variablensatz.

Es gibt zwei Gründe, aus denen es wünschenswert ist, mit einem möglichst kleinen LP zu arbeiten. Der eine ist Speicherplatz und der andere Laufzeit.
Bei großen Problemen mit vollständigen Graphen kann es unmöglich sein, alle Variablen gleichzeitig im Speicher zu halten[12].
Aber noch bevor es dazu kommt, kann die Größe des LP's die Zeit je Iteration so vergrößern, daß das Verfahren nicht mehr praktikabel ist.

Da in der Optimallösung nur ein Bruchteil der Variablen benötigt wird, sollte es doch ausreichen, wenn wir uns auf diese Variablen beschränken, das Problem lösen, und dann zeigen, daß die so gefundene Lösung optimal ist.

Also wählen wir eine geeignete Teilmenge der Variablen aus, von der wir annehmen, daß sie die Optimallösung enthält und optimieren dann das so verkleinerte Problem. Anschließend werden die reduzierten Kosten der nicht benutzten Variablen berechnet. Finden sich noch geeignete Kandidaten, nehmen wir diese auf und wiederholen die Optimierung. Andernfalls sind wir fertig.

Dieses Einbeziehen von Variablen muß dabei nicht unbedingt nach dem Beenden des Schnittebenenverfahrens für den reduzierten Variablensatz erfolgen, sondern kann auch jeweils nach

[12] z.B. bei Traveling-Salesman-Problemen mit 2000 Städten. Siehe hierzu [PR91].

einer gewissen Anzahl von Iterationen des Schnittebenenverfahrens durchgeführt werden. Die übrigen Variablen werden dann getestet, geeignete Kandidaten aufgenommen und ungeeignet erscheinende aus dem Verfahren entfernt.

Der Nachteil der Methode liegt in der Notwendigkeit, nachdem eine Lösung des reduzierten Problems gefunden wurde, möglicherweise das ganze Verfahren mit ausgetauschtem oder erweitertem Variablensatz wiederholen zu müssen.

In der vorliegenden Implementierung besteht die Möglichkeit mit reduzierten Variablen zu arbeiten. Es hat sich dabei gezeigt, daß die Reduktion des Variablensatzes auch die Degeneration des linearen Programms verringert und so zusätzlich hilft, das Lösen des LP's zu beschleunigen.

Es ist zu bedenken, daß der Erfolg der Reduktion sehr stark von der ursprünglichen Variablenauswahl abhängt. Hierzu wären noch einige Untersuchungen notwendig.
Momentan benutzen wir als Startauswahl Variablen, deren zugehörige Bögen entweder in der heuristischen Lösung enthalten sind, oder die zu den beiden kürzesten aus einem Knoten herausgehenden Bögen zählen.

Initiale Ungleichungen

Wenn wir das Schnittebenenverfahren so durchführen, wie wir es bis jetzt beschrieben haben, ist unser LP zu Beginn bis auf Variablenbeschränkungen leer.
Um den Ablauf des Verfahrens zu beschleunigen, generieren wir gültige (und von einer Nullösung natürlich verletzte) Schnittungleichungen, die dann das initiale LP bilden:

Sei r die Wurzel des Graphen D_G zu $\mathrm{ST}(D_G, T, c)$, und

$$W_i = \{v \in V \mid d(r,v) \leq i\} \text{ mit } 0 < i < \max_{t \in T}\{d(r,t)\}.$$

Füge alle sich aus den Schnitten $\delta(W_i)$ ergebenden Ungleichungen zum LP hinzu.

Der Nachteil dieser Vorgehensweise liegt in der großen Anzahl von Ungleichungen und Nichtnulleinträgen, die unter Umständen generiert werden. Deswegen haben wir das Verfahren für größere Probleme (mehr als 5000 Kanten nach der Reduktion) nicht mehr benutzt.

Kapitel 7

Ergebnisse

7.1 Die Testdaten

Immer, wenn man ein Verfahren nicht aus der Notwendigkeit heraus implementiert, ein bestimmtes Problem lösen zu müssen, stellt sich die Frage nach den Testdaten.

Bevor wir überlegen, woher wir unsere Probleme nehmen, sollten wir erst fragen, welche Anforderungen ein Problem erfüllen muß, damit es ein geeignetes Testbeispiel ist.

Um zu untersuchen, wie gut ein Verfahren ist, möchten wir am liebsten *das Steinerbaumproblem per se* haben. Das gibt es natürlich nicht, aber irgendwie sollen die Probleme typische Vertreter ihrer Klasse sein. Andererseits möchte man, um das *worstcase*- Verhalten und die Stabilität zu testen, auch die „schlimmsten Fälle“ haben.

Es läge also nahe, einen Problemgenerator zu schreiben, der zufällig eine hinreichend große Zahl von Beispielen erzeugt, damit man annehmen könnte, daß alle Fälle abgedeckt sind.

Dieser Ansatz hat leider zwei Nachteile: Zum einen haben diese generierten Probleme oft nicht das Geringste mit den in der Wirklichkeit vorkommenden Fällen zu tun. Manchmal sind sie aus diesem Grund auch viel schwerer zu lösen. Pathologische Fälle gibt es in der Praxis eher selten[1]. Andererseits erliegt man leicht der Versuchung, Probleme so zu generieren, daß sie gut zu dem benutzten Algorithmus passen.

Hinzu kommt noch, daß wenn man seine Probleme selbst erzeugt, die Vergleichbarkeit von verschiedenen Ansätzen nahezu unmöglich wird.

Der beste Ausweg aus diesem Dilemma ergibt sich, wenn mehrere Personen ihre möglicherweise unter verschiedenen Zielsetzungen erzeugten oder erstellten Probleme in einer allen zugänglichen Bibliothek sammeln.
Wie sich gezeigt hat, kommen ganz von selbst alle Arten von Beispielen zusammen, deren spezielle Problematiken mit der Zeit auch besser verstanden werden.
So können Lösungsverfahren wenigstens teilweise vergleichbar gemacht werden und man gerät nicht in Gefahr, sein Verfahren *unwissentlich* nur für eine bestimmte Klasse von Problemen zu entwerfen.

[1] Zumal unser ganzer Ansatz zur Lösung von $\mathcal{NP}$-schweren Problemen ja genau auf der Annahme beruht, daß die zu lösenden Probleme eben nicht die sind, an denen man exponentielle Komplexität besonders gut studieren kann.

Mein Dank gilt hier Ralf Borndörfer, Carlos Ferreira und Martin Grötschel, die mit der *STEINLIB* eine Sammlung von Steinerbaumproblemen erstellt haben, die über das hinausgeht, was J.E. Beasley in der OR-Library begonnen hat.

Alle im weiteren aufgeführten Problemdaten sind über die *eLib* des Konrad-Zuse-Zentrums Berlin (ZIB) verfügbar.

Datex-P:	+45050939033 (WIN) oder +2043623939033 (IXI)
Internet:	telnet elib.zib-berlin.de (170.73.108.11)
	rlogin elib.zib-berlin.de -l elib
	anonymous ftp elib.zib-berlin.de
E-Mail:	elib@zib-berlin.de, send index

Einige Informationen über die 191 insgesamt benutzten Steinerbaumprobleminstanzen lassen sich Tabelle 7.1 entnehmen. Die Spalten D, G und V geben an, ob es sich um einen dünnen,

Name	D	G	V	Z	E	Max c	Herkunft und Beschreibung	
B[01..16,18]	X			X		10	Beasley	[Bea89]
C[01..20]	X			X		10	Beasley	[Bea89]
D[01..20]	X			X		10	Beasley	[Bea89]
E[01..20]	X			X		10	Beasley	[Bea89]
R[01..46]		X		X		90	Soukoup, Chow	[SC73]
P[401..410]			X	X		1500	Chopra, Gorres, Rao	[CGR92]
P[455..466]			X		X	1500	Chopra, Gorres, Rao	[CGR92]
P[601..616]		X		X		1500	Chopra, Gorres, Rao	[CGR92]
P[619..633]		X			X	1500	Chopra, Gorres, Rao	[CGR92]
MC[2,3,13]			X	X		45	Francois Margot	
MC[7,8,11]		X		X		245	Frabcois Margot	
berlin			X		X	500	Grötschel, Kamin, Martin 52 Sehenswürdigkeiten in Berlin	
br			X		X	20000	Ferreira 58 Städte in Brasilien	
gr			X		X	20039	Borndorfer, Ferreira, Grötschel 666 Städte der Erde, 174 Hauptstädte als Terminale.	
grd.*.*		X			X	1	Martin	[Mar92]

Tabelle 7.1: Art und Herkunft der Testdaten

Gitter- oder vollständigen Graphen handelt, Z und E, ob die Kantengewichte zufällig oder mehr euklidisch (Dreiecksungleichung gilt) vergeben wurden. *Max c* gibt das größte vorkommende Kantengewicht an. Das kleinste Gewicht ist 73 bei *br* und ansonsten eins.

7.2 Die Ergebnisse

Es werden in den nachfolgenden Tabellen die Ergebnisse unserer Implementierung für die vorliegenden Testbeispiele aufgezeigt. Ein Teil der Probleminstanzen, die schon von den Reduktionstests in Kapitel 3 vollständig gelöst werden konnten, sind, ebenso wie die Testsets „B“ und „GRD“, nicht mit aufgenommen worden, da sie keine interessanten Informationen mehr bieten.

Alle nachfolgend genannten Zeiten sind CPU Sekunden, gemessen auf einer SUN 20 mit 75 MHz SuperSparc Prozessor. Die Tabellen zeigen in der Spalte *User* die vollständige Zeit die zum Ermitteln der optimalen Lösung des Problems notwendig war, inklusive Ladezeiten, Reduktionen und Heuristiken.
Dabei ist zu berücksichtigen, daß die zu einer Instanz genannte Zeit nicht notwendigerweise die schnellst mögliche ist, da nicht alle möglichen Programmparameter mit jeder Instanz durchprobiert wurden. Man kann aber annehmen, daß die gewählten Parameter eine günstige Wahl für jeweils das ganze Testset darstellen.
Alle Zeiten wurden auf einem Rechner mit derselben Version des Programms ermittelt und sind daher reproduzierbar.[2]

Die ersten Spalten $|V|$, $|E|$ und $|T|$ zeigen die Orginalgröße des Problems. Die nachfolgenden Spalten $|V|$, $|E|$ und $|T|$ zeigen die Größe der Probleminstanz die von den verwendeten Reduktionen erzielt wurden und die dann vom B&C-Verfahren gelöst wurde. Die Spalte *LP* zeigt die durch den LP-Löser, die Spalte *Sep* die durch die Separierungsalgorithmen verbrauchte Zeit an. Unter *Branch* ist die Anzahl der im Laufe des Verfahrens erzeugten Teilprobleme angegeben und unter *Iter* die Zahl der Iterationen, die das Schnittebenenverfahren insgesamt durchgeführt hat. Also wie viele LP's gelöst wurden und wie oft nach verletzten Ungleichungen gesucht wurde.

Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Br.	Iter.	LP	Sep	User	Opt.
berlin	52	1326	16	47	144	14	1	13	1	1	1	1044
br	58	1653	25	39	113	10	1	16	1	1	1	13655
gr	666	221445	174	566	3083	104	1	48	127	54	330	122467

Tabelle 7.2: B&C-Ergebnisse für Testset X

Bei Tabelle 7.2 ist der Graph *gr* bemerkenswert, der von der Kantenzahl die größte aller von uns untersuchten Probleminstanzen darstellt. Offenbar ist es gerade bei vollständigen Graphen wichtig, Reduktionen durchzuführen. Betrachtet man die Ergebnisse in [CGR92] und [Luc93], sowie die Ergebnisse für die Instanzen *mc2*, *mc3* und *mc13*, so scheinen vollständige

[2]Beim Arbeiten mit verschiedenen Betriebsystemen und LP-Löser-Versionen entsteht folgendes Problem: Da die Erzeugung von Zufallszahlen bei den verschiedenen Systemen offenbar nicht identisch implementiert ist, kommt es zu Unterschieden, immer wenn wir mit Perturbationen arbeiten. Ebenso können verschiedene Versionen von CPLEX unterschiedliche Lösungen für ein lineares Programm liefern. Zwar ist der Zielfunktionswert gleich, da wir aber die Lösung benutzen, um die Primalheuristik zu beeinflussen, gibt es möglicherweise Auswirkungen auf den Zeitpunkt, zu dem die Heuristik eine bestimmte obere Schranke ermittelt. Und das hat mitunter einen wesentlichen Einfluß auf die Dauer des Verfahrens.

Graphen zu den am schwersten zu lösenden Instanzen zu gehören. Alle drei Probleminstanzen wurden mit zulässiger Perturbation und initialen Ungleichungen gelöst.

Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Br.	Iter.	LP	Sep	User	Opt.
mc2	120	7140	60	120	478	60	7	39	14	7	31	71
mc3	97	4656	45	97	1203	45	13	83	196	19	235	47
mc13	150	11175	80	149	623	80	429	1224	326	232	1104	92
mc7	400	760	170	51	81	27	1	5	1	1	7	3417
mc8	400	760	188	56	85	33	1	9	1	1	7	1566
mc11	400	760	213	40	58	26	1	6	1	1	5	11689

Tabelle 7.3: B&C-Ergebnisse für Testset MC

In Tabelle 7.3 sticht *mc13* hervor. Das Schnittebenenverfahren konnte nach 27 Sekunden und 15 Iterationen, bei einem LP-Zielfunktionswert von 90.2266, keine weiteren verletzten Ungleichungen mehr separieren. Zu diesem Zeitpunkt lag die durch die Heuristik gefundene obere Schranke bei 94. Der Optimalwert wurde von der Heuristik beim neunten untersuchten Teilproblem nach acht Fixierungen auf eins gefunden. Maximal mußten 16 Variablen fixiert werden, um ein Teilproblem abschließen zu können. Daran, daß der Optimalwert sehr früh gefunden wurde, zeigt sich, daß die Tiefensuche hier eine geeignete Auswahlstrategie für die Teilprobleme war. Andererseits wird auch klar, daß es wenig geholfen hätte, wenn die Heuristik schon früher eine optimale Lösung gefunden hätte.[3] Alle Probleminstanzen wurden mit zulässiger Perturbation und initialen Ungleichungen gelöst.
Die in den Tabellen 7.4, 7.5 und 7.6 gezeigten Ergebnisse geben einen Hinweis darauf, wie sich die Lösungszeiten verändern, wenn die Instanzen größer werden und die Anzahl der Terminale sich ändert. Da keine initialen Ungleichungen verwendet wurden, sieht man sehr deutlich, daß bei einer geringen Zahl von Terminalen oft eine sehr hohe Anzahl von Iterationen durchgeführt wurde (*c11, c17, d6, d11, d17, e7, e11, e12, e16* und *e17*). Dies ist nicht verwunderlich, wenn man das Separierungsverfahren betrachtet, bei dem je Iteration maximal $|T|$ verletzte Ungleichungen gefunden werden können. Als Folge der geringen Veränderung des zugehörigen linearen Programms, wird das duale Simplexverfahren extrem schnell durchgeführt.
Bei *d17* werden maximal 47, meistens unter 15, Simplexiterationen gebraucht, um ein neues Optimum für das LP zu finden. Das erklärt das Mißverhältnis zwischen den Zeiten für *d17* und *d18*. Dabei wird auch deutlich, daß die Anzahl der Variablen weniger ausschlaggebend für die Lösungszeit ist als das Verhältnis von Terminalen zu Nicht-Terminalen.
Die Diagramme 7.1 und 7.2 zeigen, wie sich die Schranken im Verlauf des Verfahrens annähern. Die Anzahl der Nicht-Null-Einträge ist mit 1/600 bzw. 1/1200 skaliert und fast immer monoton steigend. Wann immer ein Absinken beobachtet werden konnte, blieb bei diesen Iterationen der Zielfunktionswert des LP's weitestgehend konstant.

Problem *e18* stellt einen Sonderfall dar. Es ist nach Kenntnisstand des Autors das letzte der von Beasley in [Bea89] eingeführten Probleme, für das der Optimalwert noch nicht bekannt

[3] Bei *e18* trifft dies nicht so zu.

Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Br.	Iter.	LP	Sep	User	Opt.
c01	500	625	5	138	247	5	1	23	1	1	1	85
c02	500	625	10	122	221	8	1	29	1	1	1	144
c03	500	625	83	87	148	35	1	8	1	1	1	754
c04	500	625	125	81	129	36	1	10	1	1	1	1079
c05	500	625	250	23	36	15	1	6	1	1	1	1579
c06	500	1000	5	368	839	5	1	48	1	2	4	55
c07	500	1000	10	380	856	9	1	32	2	2	4	102
c08	500	1000	83	322	656	58	1	26	5	7	14	509
c09	500	1000	125	267	503	77	1	21	4	5	11	707
c10	500	1000	250	81	131	51	1	7	1	1	1	1093
c11	500	2500	5	498	2045	5	1	114	7	9	19	32
c12	500	2500	10	492	1786	9	1	79	5	7	16	46
c13	500	2500	83	399	945	65	1	30	10	12	26	258
c14	500	2500	125	290	591	72	1	15	2	5	10	323
c15	500	2500	250	147	257	70	1	9	1	1	3	556
c16	500	12500	5	500	3504	5	1	23	2	4	14	11
c17	500	12500	10	498	2970	8	1	76	7	12	26	18
c18	500	12500	83	452	1362	57	1	26	12	12	31	113
c19	500	12500	125	355	958	41	1	22	2	2	9	146
c20	500	12500	250	16	30	6	1	3	1	1	4	267

Tabelle 7.4: B&C-Ergebnisse für Testset C

Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Br.	Iter.	LP	Sep	User	Opt.
d01	1000	1250	5	273	507	5	1	42	1	1	3	106
d02	1000	1250	10	284	521	10	1	28	1	1	2	220
d03	1000	1250	167	162	265	67	1	9	1	1	2	1565
d04	1000	1250	250	102	162	53	1	10	1	1	2	1935
d05	1000	1250	500	42	66	29	1	5	1	1	1	3250
d06	1000	2000	5	761	1738	5	1	129	10	9	22	67
d07	1000	2000	10	747	1708	10	1	58	3	5	10	103
d08	1000	2000	167	622	1247	127	1	20	9	19	40	1072
d09	1000	2000	250	482	904	159	1	15	6	17	32	1448
d10	1000	2000	500	165	283	99	1	8	1	1	5	2110
d11	1000	5000	5	991	4390	5	1	168	17	28	59	29
d12	1000	5000	10	995	3829	9	1	114	14	24	53	42
d13	1000	5000	167	804	1866	130	1	20	9	27	55	500
d14	1000	5000	250	653	1381	167	1	25	17	42	86	667
d15	1000	5000	500	327	587	154	1	11	3	5	18	1116
d16	1000	25000	5	1000	8621	5	1	69	11	29	104	13
d17	1000	25000	10	999	8031	9	1	135	34	68	169	23
d18	1000	25000	167	910	2898	112	1	35	54	92	194	223
d19	1000	25000	250	810	2386	127	1	20	12	42	95	310
d20	1000	25000	500	88	169	30	1	4	1	1	22	537

Tabelle 7.5: B&C-Ergebnisse für Testset D

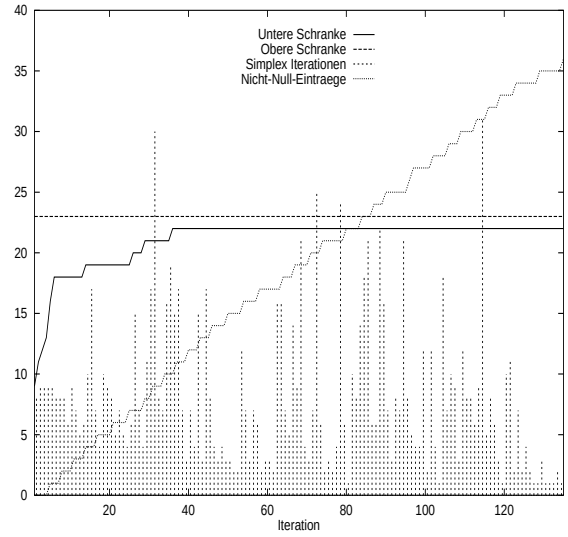


Abbildung 7.1: Schranken bei *d17*

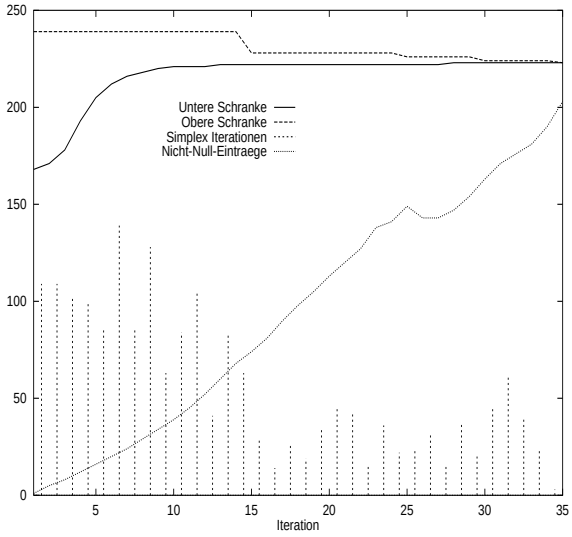


Abbildung 7.2: Schranken bei *d18*

war. Wie erwartet, ist es auch das Problem, das der hier vorgestellten Implementierung am längsten widerstanden hat und dessen Lösung die meiste Rechenzeit verbraucht.

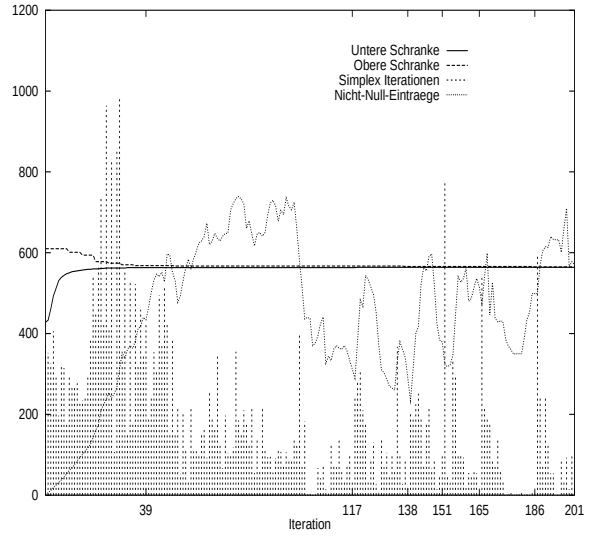


Abbildung 7.3: Schranken bei *e18*

Interessanterweise sind die in [CGR92] und [Luc93] genannten unteren Schranken 563.003 bzw. 563.5 ausreichend, um das Problem zu lösen, wenn es gelingt, den Optimalwert mit der Primalheuristik zu ermitteln. Im hier vorgestellten Fall wird eine untere Schranke von 563.015 bereits nach 39 Iterationen des Schnittebenenverfahrens erreicht. Gelänge es der Primalheuristik bis zu diesem Zeitpunkt eine optimale Lösung zu finden, könnte das Problem in einem Bruchteil der Zeit gelöst werden.
Es läßt sich schließen, das *e18* vor allem eine Herausforderung für die Heuristiken darstellt.

Wie man den Diagrammen 7.1, 7.2 und 7.3[4] entnehmen kann, ist die Zahl der Simplexiterationen weitgehend unabhängig von der Anzahl der Nicht-Null-Einträge. Da wir durchgehend das duale Simplexverfahren benutzen, läßt sich, wie erwartet, ein Ansteigen der Iterationsanzahl nach einer Fixierung beobachten, da hier die Basis des LP's nicht mehr dual zulässig ist.[5]

[4]Nicht-Null-Einträge sind mit 1/1000 skaliert.

[5]Die sporadische Untersuchung der Programmlaufprotokolle legt nahe, daß ein starker Zusammenhang zwischen der Anzahl der hinzugefügten Ungleichungen und der Anzahl der Simplexiterationen besteht. Das ist erklärbar, da die Anzahl der Iterationen eines Simplexalgorithmus unter anderem davon abhängt, wie

Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Br.	Iter.	LP	Sep	User	Opt.
e01	2500	3125	5	678	1282	5	1	51	2	4	7	111
e02	2500	3125	10	705	1313	9	1	71	4	8	13	214
e03	2500	3125	417	397	666	175	1	11	3	8	20	4013
e04	2500	3125	625	241	395	137	1	10	1	2	8	5101
e05	2500	3125	1250	64	107	47	1	7	1	1	3	8128
e06	2500	5000	5	1845	4315	5	1	86	7	15	29	73
e07	2500	5000	10	1889	4364	10	1	129	39	43	92	145
e08	2500	5000	417	1548	3138	309	1	25	118	292	679	2640
e09	2500	5000	625	1180	2222	359	1	18	30	108	340	3604
e10	2500	5000	1250	468	801	262	1	17	10	35	83	5600
e11	2500	12500	5	2498	11868	5	1	181	38	89	189	34
e12	2500	12500	10	2498	11393	10	1	138	97	152	331	67
e13	2500	12500	417	2032	4779	320	1	40	634	763	1908	1280
e14	2500	12500	625	1631	3480	372	1	20	51	174	614	1732
e15	2500	12500	1250	643	1161	318	1	20	20	105	240	2784
e16	2500	62500	5	2500	25184	5	1	275	120	366	1207	15
e17	2500	62500	10	2500	21508	10	1	161	101	351	1022	25
e18	2500	66250	417	2224	5995	394	11	207	25870	18220	53982	564
e19	2500	62500	625	1793	4832	244	1	33	51	92	588	758
e20	2500	62500	1250	264	531	79	1	7	1	1	249	1342

Tabelle 7.6: B&C-Ergebnisse für Testset E

Abarbeitung der Teilprobleme bei *e18*						
Teilproblem	1	2	3	4	5	6
Iterationen	117	21	13	14	21	15
Untere Schranke	563.459	563.600	563.620	563.744	563.822	563.983
Oberer Schranke	567	566	566	566	565	564
Fixierungen	0	1	2	3	4	5

Es sei noch auf *e16* hingewiesen. Nach 79 Iterationen des Schnittebenenverfahrens wird eine untere Schranke von 14 erreicht. Die Lösung der LP-Relaxierung behält dann diesen Wert für 195 Iterationen bei.
Auffällig an *r34*, *r35* und *r45* ist das Verhältnis zwischen der von LP-Löser verbrauchten Zeit und der Separierungszeit.
Die Untersuchung des Ablaufprotokolls bei *r45* zeigte keine numerischen Probleme beim LP-Löser[6], aber eine, im Vergleich zu *e4*, hohe Anzahl von durchschnittlich 200 Simplexiterationen je Reoptimierung. Sowohl in [CGR92] wie auch in [Luc93] werden Steinerbaumprobleme auf Gittergraphen als schwerer lösbar als vergleichbar große Probleme auf anderen Graphen

stark sich die Startbasis von der Basis der optimalen Lösung unterscheidet. Das ist auch bei Versuchen zum Steinerbaumpackungsproblem deutlich geworden. Da hier das Hinzufügen von Ungleichungen oft eine sehr starke Änderung der Lösung verlangt, war hier die Verwendung der bisherigen Lösung als Startbasis nahezu nutzlos. Ein Punkt weitergehender Untersuchungen wäre der Zusammenhang mit der Änderung des Zielfunktionswertes. Möglicherweise könnte nur das Hinzufügen der am stärksten von der LP-Lösung verletzten Ungleichungen den Ablauf des Simplexverfahrens beschleunigen, ohne daß die Anzahl der Iterationen des Schnittebenenverfahrens stark ansteigt.

[6] Durchgängig nur eine Basisrefaktorisierung je 60 Simplexiterationen.

Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Br.	Iter.	LP	Sep	User	Opt.
r04	64	112	8	28	44	8	1	10	1	1	1	254
r08	24	37	12	8	9	6	1	1	1	1	1	236
r13	42	71	9	24	37	8	1	7	1	1	1	150
r15	100	180	14	34	56	11	1	5	1	1	1	148
r17	48	82	10	9	14	4	1	4	1	1	1	200
r18	182	337	62	119	203	42	1	11	2	1	3	404
r19	168	310	14	21	35	6	1	8	1	1	2	188
r30	28	45	12	8	10	5	1	1	1	1	1	110
r31	130	237	14	65	111	14	1	11	1	1	2	259
r32	210	391	19	123	221	19	1	25	7	1	10	313
r33	132	241	18	75	136	11	1	14	2	1	3	268
r34	272	511	19	175	321	19	1	30	22	2	29	241
r35	240	449	18	123	218	18	1	25	7	1	12	151
r38	100	180	14	61	105	12	1	13	1	1	2	166
r39	100	180	14	51	89	10	1	11	1	1	1	166
r40	64	112	10	36	59	10	1	13	1	1	1	155
r41	144	263	20	99	175	16	1	14	3	1	4	224
r42	81	144	15	22	32	10	1	6	1	1	1	153
r43	195	362	16	127	227	16	1	26	8	1	12	255
r44	196	364	17	137	246	17	1	32	11	2	15	252
r45	270	507	19	196	360	19	1	48	46	4	59	220

Tabelle 7.7: B&C-Ergebnisse für Testset R

angesehen, da sie ein hohes Maß an Symmetrie aufweisen. Wenn das aus den hier gezeigten Ergebnissen nur eingeschränkt ersichtlich ist, dann vermutlich weil die Reduktionen große Teile der Symmetrie zerstören.
Es ist anzumerken, daß obwohl, wie wir in Kapitel 4 gesehen haben, die Primalheuristiken auf Gittergraphen gute Ergebnisse liefern, es bei *r45* der Heuristik nicht gelang, eine optimale Lösung zu finden.

Name	$\|V\|$	$\|E\|$	$\|T\|$	$\|V\|$	$\|E\|$	$\|T\|$	Br.	Iter.	LP	Sep	User	Opt.
p401	100	4950	5	84	185	5	1	27	1	1	1	155
p402	100	4950	5	68	142	5	1	8	1	1	1	116
p403	100	4950	5	87	198	5	1	31	1	1	1	179
p405	100	4950	10	65	124	9	1	8	1	1	1	270
p406	100	4950	10	82	173	8	1	16	1	1	1	290
p407	100	4950	20	72	142	14	1	15	1	1	1	590
p409	100	4950	50	13	19	10	1	5	1	1	1	963
p455	100	4950	5	100	1057	5	1	61	4	4	9	1138
p456	100	4950	5	100	880	5	1	79	4	4	9	1228
p457	100	4950	10	97	652	8	1	52	4	2	6	1609
p458	100	4950	10	99	593	9	1	36	4	2	6	1868
p459	100	4950	20	94	412	16	1	18	1	1	2	2345
p460	100	4950	20	94	446	17	1	26	3	1	4	2959
p461	100	4950	50	58	127	22	1	11	1	1	1	4474
p463	200	19900	10	200	2214	10	1	77	36	15	55	1510
p464	200	19900	20	191	1754	14	1	97	61	15	80	2545
p465	200	19900	40	183	799	34	1	31	7	4	14	3853
p466	200	19900	100	120	278	45	1	20	1	1	5	6234
p601	100	180	5	77	134	5	1	22	1	1	1	10230
p602	100	180	5	77	133	5	1	21	1	1	1	8083
p603	100	180	5	77	135	4	1	17	1	1	1	5022
p604	100	180	10	70	122	7	1	19	1	1	1	11397
p605	100	180	10	70	120	7	1	15	1	1	1	10355
p606	100	180	10	81	137	9	1	20	1	1	1	13048
p609	100	180	20	68	114	14	1	12	1	1	1	18263
p610	100	180	50	26	46	9	1	9	1	1	1	30161
p612	100	180	50	29	44	12	1	10	1	1	1	30258
p614	200	370	20	179	307	18	1	27	2	1	3	27276
p615	200	370	40	145	244	33	1	27	2	1	3	42474
p616	200	370	100	43	67	25	1	13	1	1	1	62263
p619	100	180	5	87	162	5	1	27	1	1	1	7485
p620	100	180	5	85	160	4	1	31	1	1	1	8746
p622	100	180	10	86	159	8	1	25	1	1	1	15972
p623	100	180	10	87	158	10	1	30	1	1	2	19496
p624	100	180	20	77	138	11	1	24	1	1	1	20246
p625	100	180	20	81	150	15	1	18	1	1	1	23078
p627	100	180	50	39	67	16	1	10	1	1	1	40647
p630	200	370	10	187	353	8	1	45	3	2	4	26125
p631	200	370	20	185	343	19	1	35	2	1	3	39067
p632	200	370	40	171	318	26	1	26	3	2	5	56217
p633	200	370	100	66	118	23	1	13	1	1	1	86268

Tabelle 7.8: B&C-Ergebnisse für Testset P

7.3 Ein kleiner Vergleich

Jeder Vergleich von Verfahren, wenn diese auf verschiedenen Rechnern mit unterschiedlichen zugrundeliegenden Softwarepaketen (hier LP-Lösern) ausgeführt wurden, muß zweifelhaft bleiben.

Jedes Programm kann noch verbessert werden, jeder Algorithmus verfeinert, und für alle Probleme gibt es eine CPU, die geeigneter ist, sie zu lösen. Dennoch wird niemand, der um eine einfache und schnelle Möglichkeit weiß, sein Verfahren zu beschleunigen, diese nicht nutzen.

Und so zeigen alle Ergebnisse, zumindest tendenziell, was möglich war und was nicht. Tabelle 7.9 zeigt die Summe der Zeiten für die Testsets „C“, „D“, „E“ und „R“, die Lucena in [Luc93] für eine Implementierung des ungerichteten Modells mit Knotenvariablen in Verbindung mit einer Lagrangerelaxierung angibt, sowie die von Chopra, Rao und Gorres in [CGR92] erreichten Zeiten für ihre Implementierung des gerichteten Modells.

	1	2	3
$\sum$ Test Set	C.G.R. Vax 8700 XMP 1992	Lucena Sparc 10 XMP[7] 1993	Koch Sparc 20 CPLEX 1995
C	61417[8]	1950/48	268
D	285195[9]	7799/63	974
E[10]	599599	168893/6911	7622
R	—[11]	256714/256292	200

Tabelle 7.9: Vergleich

Die angegebenen Zeiten verstehen sich als CPU-Sekunden auf der jeweiligen Maschine. Dabei ist zu bedenken, daß diese Summen nur ein sehr eingeschränktes Bild auf die Leistung der Verfahren werfen.

Es wenig sinnvoll, die Geschwindigkeiten der jeweiligen Maschinen und LP-Löser zu vergleichen, aber es ist zu erkennen, daß im Laufe der Zeit Fortschritte erzielt wurden, die nicht allein mit der Leistungssteigerung der Hardware zu erklären sind.

[7] Die erste Zahl in der Spalte gibt die Gesamtzeit an, und die zweite den Anteil des Schnittebenenverfahrens. Wie man sieht, werden die Testsets C, D und E nahezu ausschließlich mit der Lagrangerelaxierung gelöst. Daher ist hier der verwendete LP-Löser nicht von Bedeutung.

[8] Summe wird von Problem C18 mit 45848 s dominiert.

[9] Summe wird von Problem D18 mit 245192 s dominiert.

[10] Ohne Problem *e18*.

[11] Die Daten lagen nicht vor.

7.4 Schluß

Am Ende bleibt immer ein Algorithmus übrig, der nicht untersucht ist, ein Trick übrig der noch nicht implementiert ist.

Wir haben in dieser Arbeit gezeigt, daß ein Schnittebenenverfahren auf Grundlage eines bidirektionalen Modells ein geeignetes Mittel ist, eine untere Schranke für den Optimalwert eines Steinerbaumproblems in Graphen zu bestimmen und daß wirksame Reduktionen einen wichtigen Teil des Lösungsverfahrens darstellen.

Es scheint, daß die größten Möglichkeiten für eine weitere Beschleunigung des Verfahrens in einer Verbesserung der Primalheuristiken liegen.

Ich möchte mich an dieser Stelle bei allen bedanken, die mir durch Diskussion, Anregungen oder Durchsicht geholfen haben. Insbesondere Monica Ahuna, Claudia Betz-Haubold, Ralf Borndörfer, Carlos Ferreira, Alexander Martin, Christian Müller, Christian Schmidt, Tuomo Takkula, Roland Wunderling und Ines Höschel.

Anhang A

Zwei Verallgemeinerungen

In diesem Anhang sollen noch zwei Verallgemeinerungen des Steinerbaumproblems in Graphen kurz vorgestellt werden. Wir werden dann Hinweise auf die Änderungen geben, die notwendig sind um unser Verfahren für diese Probleme anzupassen.

A.1 Das Steinerbaumproblem mit Knotengewichten

Ordnen wir in einem Steinerbaumproblem $\mathrm{ST}(G, T, c)$ jedem Knoten $v \in V$ noch ein Knotengewicht c_v zu und fordern für eine optimale Lösung S, daß ihr Gewicht

$$c(S) = \sum_{e \in S} c_e + \sum_{v \in V(S)} c_v$$

minimal ist, so erhalten wir das *Steinerbaumproblem mit Knotengewichten in Graphen.*

Da die Terminale ohnehin in $V(S)$ enthalten sein müssen, können wir $c_v := 0$ für alle $v \in T$ setzen und später $\sum_{v \in T} c_v$ zum Gewicht der optimalen Lösung hinzuaddieren.

Modifizieren wir unser gerichtetes Modell jetzt, indem wir $c_a := c_a + c_v$ für alle $a \in \delta_v^-$ für alle $v \in N$ setzen, so ist jede gewichtsminimale Lösung, die wir erhalten, auch gewichtsminimal für das Steinerbaumproblem mit Knotengewichten.

A.2 Das Packen von Steinerbäumen

Eine Verallgemeinerung des Steinerbaumproblems, bei der in einem Graphen nicht nur ein, sondern mehrere disjunkte Steinerbäume gesucht werden, ist das *gewichtete Steinerbaumpackungsproblem in Graphen* (STP).

Wir werden im Folgenden auf Aussagen und Ergebnisse aus [GMW92b], [GMW92a] und [Mar92] zurückgreifen, ohne gesondert darauf hinzuweisen.

Eine mögliche Formulierung lautet:

> Gegeben sei ein Graph $G = (V, E)$ mit Kantengewichte $c_e \in \mathbb{R}_+$ und Kantenkapazitäten $u_e \in \mathbb{N}$ für alle $e \in E$, sowie eine nicht-leere *Netzliste* $L = \{T_1, \ldots, T_{|L|}\}$ mit Terminalmengen $T_k \subseteq V$ für $k = 1, \ldots, |L|$.
> Finde für jede Terminalmenge $T_k \in L$ eine zulässige Lösung $S_k \subseteq E$ des Steinerbaumproblems $\mathrm{ST}(G, T_k, c)$, so daß $\sum_{k=1}^{|L|} c(S_k)$ minimal ist und daß $\sum_{k=1}^{|L|} |S_k \cap \{e\}| \leq u_e$ für alle $e \in E$ gilt.

Das Problem besteht also darin, mehrere Steinerbäume gewichtsminimal in einen Graphen zu packen, ohne dabei die Kapazitäten der Kanten zu überschreiten.

Wie man leicht sieht, ist das Steinerbaumpackungsproblem $\mathrm{ST}(G, L, c, u)$ für den Fall $|L| = 1$ genau das Steinerbaumproblem.

Wir können der einfachheithalber davon ausgehen, daß die Terminalmengen disjunkt sind, d.h. $T_i \cap T_k = \emptyset$ für alle $i \neq k$, $i, k \in \{1, \ldots, |L|\}$. Das hat für das Problem keine wesentliche Bedeutung, da sich jedes Problem mit nicht disjunkten Terminalmengen trivial durch Einführung eines weiteren Knotens und einer Kante, für jede Überschneidung der Terminalmengen, in eines mit disjunkten Mengen überführen läßt.

Benutzen wir wieder einen bidirektionalen Graphen $D_G = B(G) = (V, A)$, so müssen wir eine Wurzel $r_k \in T_k$ für jede Terminalmenge T_k mit $k = 1, \ldots, |L|$ beliebig wählen. Wie bei den Bogengewichten, seien die Bogenkapazitäten u_a für $a \in A$ entsprechend denen der induzierenden Kanten.

Eine Modellierung von $\mathrm{ST}(D_G, L, c, u)$ basierend auf dem gerichtete Modell aus Kapitel 2 sieht dann wie folgt aus

$$\min \sum_{k=1}^{|L|} \sum_{a \in A} x_a^k c_a$$

$$\text{s.t.:} \quad \sum_{a \in \delta^-(W_k)} x_a^k \geq 1, \quad \begin{array}{l}\text{für alle } W_k \subseteq V \setminus \{r_k\},\ W_k \cap T_k \neq \emptyset \\ \text{und } k = 1, \ldots, |L|\end{array} \tag{A.1}$$

$$\sum_{k=1}^{|L|} x_a^k \leq u_a, \text{ für alle } a \in A \tag{A.2}$$

$$x_a^k \in \{0, 1\}, \tag{A.3}$$

wenn wir binäre Variablen x_a^k, $a \in A$, $k = 1, \ldots, |L|$, sowie *Kapazitätsungleichungen* (A.2) einführen.

Für jedes Netz $T_k \in L$ ist dann die Variable

$$x_e^k = \begin{cases} 1, & \text{falls die Kante } e \text{ in der Lösung } S_k \text{ zur Terminalmenge } T_k \text{ verwendet wird,} \\ 0, & \text{sonst.} \end{cases}$$

Es lassen sich viele Erkenntnisse für das Steinerbaumproblem auf das Steinerbaumpackungsproblem übertragen, wenn man zusätzlich die Kantenkapazitäten berücksichtigt.

Diese Kantenkapazitäten machen aber gerade den entscheidenden Unterschied zwischen den beiden Problemen aus. Man kann beim STP nicht mehr trivial eine zulässige Lösung finden. Tatsächlich ist schon das in den meisten Fällen $\mathcal{NP}$-vollständig.

Für unser Verfahren bedeutet dies, daß wir nicht mehr davon ausgehen können, daß uns jederzeit eine zulässige Lösung zur Verfügung steht. Und wir müssen berücksichtigen, daß es möglicherweise keine gibt.

Auch die Reduktionsverfahren, die wir in Kapitel 3 vorgestellt haben, lassen sich ausnahmslos nicht auf das Steinerbaumpackungsproblem anwenden.

Wir haben einige Experimente mit der LP-Relaxierung der obigen Modellierung gemacht und dabei folgendes festgestellt:

- Das Fehlen einer leistungsfähigen Primalheuristik stellt ein großes Problem dar.
- Die Differenz zwischen den Optima der LP-Relaxierung und des binären Programms ist unserer Erfahrung nach größer als beim Steinerbaumproblem. Hier wäre es notwendig, weitere der in [Mar92] beschriebenen Klassen von Ungleichungen zu separieren.
- Die Entartung des LP's ist stärker als beim Steinerbaumproblem und das Hinzufügen von Ungleichungen, insbesondere Kapazitätsungleichungen führt zu sich stark verändernden LP-Lösungen. Als Folge davon steigt die Zahl der zur Reoptimierung benötigten Iterationen des dualen Simplexverfahrens stark an.
- Da die Zahl der null-wertigen Variablen ungleich größer ist, als beim Steinerbaumproblem, bietet sich ein Arbeiten auf einem reduzierten Variablensatz an. Hier wäre eine zulässige Lösung eine große Hilfe beim Treffen einer geeigneten Startauswahl.

Es ist zwar gelungen, kleinere Beispiele zu lösen, aber ohne weitere Separierer und vor allem eine zufriedenstellende Primalheuristik ist unser Verfahren noch ungeeignet dieses Problem zu lösen.

Anhang B

Das Programm

Hier wollen wir einen kleinen Einblick in den Aufbau, Umfang und Aufruf des Programms geben.

Für alle Testläufe wurde das Programm mit dem GCC Version 2.7.0 und den Optionen `-O3 -fomit-frame-pointer -funroll-loops -DNDEBUG` compiliert. Unter Solaris zusätzlich mit `-msupersparc`, unter Linux und SCO Unix mit `-m486`.

Name	Zeilen	Inhalt
display.c	400	Graphische Ausgabe unter X11
dofig.c	133	Ausgabe als *xfig* Grafik
evaluate.c	576	Kern des Schnittebenenverfahrens
fixlogic.c	283	Fixierungslogik (kritische Schnitte)
graph.h	126	Definitionen für die Graphlib Routinen
graphlib.c	1054	Routinen zum Bearbeiten von Graphen
heurist.c	451	Kürzeste-Wege-Heuristik
heurist2.c	451	Durschnittliche-Entfernungs-Heuristik
jack3.c	328	Hauptprogramm
jack3.h	276	Strukturdefinitionen, Konstanten
load.c	515	Einlesen der Probleme
mincut.c	1080	Berechnen eines minimalen Schnittes
mshell.c	378	Routinen zur Feststellung von Speicherzuordnungsfehlern
mshell.h	54	Definitionen zur Überlagerung von *malloc()* & Co.
pool.c	417	Verwaltung der Ungleichungen
portab.h	46	Portabilitätsdefinitionen
reduce.c	1544	Reduktionstests für Steinerbaumprobleme Teil 1
save.c	87	Speichern eines Problems
sdtest.c	748	Reduktionstests für Steinerbaumprobleme Teil 2
select.c	170	Auswahl der Variablen bei reduziertem Variablensatz
separate.c	170	Code für die Separierer
solve.c	733	Branch and Bound Verfahren
statist.c	149	Statistiken zu Graphen
stpath.c	297	Dijkstras Algorithmus zur Berechnung kürzester Wege
support.c	225	Variablenfixierungen etc.
timer2.c	115	Zeitmessung
validate.c	178	Zulässigkeit einer Lösung überprüfen
version.c	32	Versionskontrolle
Summe	11676	Zuviel zum Abdrucken

Tabelle B.1: Übersicht über die Quelldateien

NAME

Jack the third - Branch & Cut–Steinerbaumproblemlöser.

SYNTAX

jack33 [option] *Dateiname*

OPTIONEN

Die Optionen zum Programmaufruf müssen durch Leerzeichen getrennt werden und die Parameter müssen den Optionen immer direkt und ohne Leerzeichen folgen. Der jeweils unterstrichene Eintrag gibt die Voreinstellung des Programms an.

-b{0,1} Schaltet die rückwärtigen Schnitte (Backcuts) aus bzw. ein.

-c{0,1} Schaltet den „Creepflow“ beim Separieren aus bzw. ein.

-d{0,1} Schaltet das sofortige Herausnehmen einer nicht straff erfüllten Ungleichung aus bzw. ein.

-e Speichert das Problem nach der Reduktion als *Dateiname.red* in Beasleys Format ab.

-f{0,1} Schaltet die Separierung der Flußungleichungen aus bzw. ein.

-g Schaltet die graphische Ausgabe ein.

-l{0,1} Schaltet die Fixierungslogik vor Beginn des Schnittebenenverfahrens aus bzw. ein. (Z.Zt. die Suche nach kritischen Schnitten beim Steinerbaumpackungsproblem.)

-m{0,1} Legt fest, ob Speicherplatz auf Kosten der Geschwindigkeit gespart werden soll. Bei 0 wird der zum Separieren benötigte Speicher einmalig alloziert und erst am Ende des Programms wieder freigegeben, bei 1 wird jedesmal vor dem Aufruf des LP-Lösers der Speicher dealloziert.

-n{0,1} Schaltet die verschachtelten Schnitte (nested Cuts) beim Separieren aus bzw. ein.

-p{0,1,2} Das Problem wird nicht / zulässige / unzulässig perturbiert.

-r{0,...,3,4} Legt verschieden Reduktionsabläufe fest. Bei 0 wird nicht reduziert, 3 ist die „schnelle“ Reduktion und 4 die „ausgiebige“.

-s*Num* Speichert das LP nach Iteration *Num*.

-v{0,1} Legt fest, ob mit allen Variablen gearbeitet werden soll. Bei 1 wird das Problem mit einem reduzierten Variablensatz gelöst.

-w{0,1,2} Bei 0 wird ein initiales LP erstellt, wenn das Problem weniger als 10000 Variablen hat. Bei 1 wird nie, bei 2 immer ein initiales LP generiert.

-x*Pfad* Schreibt nach jeder Iteration die LP-Lösung in die Datei *Pfad.Dateiname.Node.Iteration*.

Literaturverzeichnis

[Abr94] Michael Abrash. *Zen of Code Optimization*. Coriolis Group, 1994.

[AMO92] Ravindra K. Ahuja, Thomas Magnanti, and James B. Orlin. *Network Flows*. Prentice Hall, 1992.

[AMOT88] Ravindra K. Ahuja, Kurt Mehlhorn, James B. Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. Technical Report 193, MIT, Operations Research Center, 1988.

[Ane80] Y. P. Aneja. An integer linear programming approach to the steiner problem in graphs. *Networks*, 10:167–178, 1980.

[AS90] Pankaj Kumar Agarwal and Man-Tak Shing. Algorithms for special cases of rectilinear steiner trees: I. points on the boundary of a rectilinear rectangle. *Networks*, 20:453–485, 1990.

[Bea84] J. E. Beasley. An algorithm for the steiner problem in graphs. *Networks*, 14:147–159, 1984.

[Bea89] J. E. Beasley. An sst-based algorithm for the steiner problem in graphs. *Networks*, 19:1–16, 1989.

[Ben86] Jon Bentley. *Programming Perls*. Addison-Wesley, 1986.

[Ber90] Marshall Bern. Faster exact algorithms for steiner trees in planar networks. *Networks*, 20:109–120, 1990.

[Ber91] Dimitri P. Bertsekas. *Linear Network Optimization: Algorithms and Codes*. The MIT Press, 1991.

[Bix92] Robert E. Bixby. Das Implementieren des Simplex-Verfahrens: Die Startbasis. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1992. Preprint SC 92-11.

[BM94] Djangir A. Babayev and Sahib S. Mardanov. Reducing the number of variables in integer and linear programming problems. *Computational Optimization and Applications*, 3:99–109, 1994.

[BP87] Anantaram Balakrishnan and Nitin R. Patel. Problem reduction methods and a tree generation algorithm for the steiner network problem. *Networks*, 17:65–85, 1987.

[CGR92] Sunil Chopra, Edgar R. Gorres, and M. R. Rao. Solving the steiner tree problem on a graph using branch and cut. *ORSA Journal on Computing*, 4(3):320–335, 1992.

[Chv83] Vašek Chvátal. *Linear Programming*. W. H. Freeman and Company, 1983.

[CR88a] Sunil Chopra and M. R. Rao. The steiner tree problem I: Formulations, compositions and extension of facets. Report 88-82, New York University, 1988.

[CR88b] Sunil Chopra and M. R. Rao. The steiner tree problem II: Properties and classes of facets. Report 88-83, New York University, 1988.

[CR94a] Sunil Chopra and M. R. Rao. The steiner tree problem I: Formulations, compositions and extension of facets. *Mathematical Programming 64*, pages 209–229, 1994.

[CR94b] Sunil Chopra and M. R. Rao. The steiner tree problem II: Properties and classes of facets. *Mathematical Programming 64*, pages 231–246, 1994.

[DV89a] C. W. Duin and A. Volgenant. An edge elimination test for the steiner problem in graphs. *Operation Research Letters*, 8:79–83, 1989.

[DV89b] C. W. Duin and A. Volgenant. Reduction tests for the steiner problem in graphs. *Networks*, 19:549–567, 1989.

[DW71] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1:195–207, 1971.

[DZ92] Ding-Zhu Du and Yanjun Zhang. On better heuristics for steiner minimum trees. *Mathematical Programming*, 57:193–202, 1992.

[EK72] J. Edmonds and R. M. Karp. Theoretical improvement in algorithmic efficiency for network flow problems. *J. ACM 19*, pages 248–264, 1972.

[GGL94] R. Graham, M. Grötschel, and L. Lovász. *Handbook on Combinatorics*. North Holland, 1994.

[GH88] M. Grötschel and O. Holland. Solution of large-scale symmetric travelling salesman problems. Technical report, Institut für Matehmatik, Universität Augsburg, 1988. Report 73.

[GLS88] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[GM90] Martin Grötschel and Clyde L. Monma. Integer polyhedra arising from certain network design problems with connectivity constraints. *SIAM J. Disc. Math.*, 3:502–523, 1990.

[GM93] Michel X. Goemans and Young-soo Myung. A catalog of steiner tree formulations. *Networks*, 23:19–28, 1993.

[GMW92a] M. Grötschel, A. Martin, and R. Weismantel. Packing steiner trees: A cutting plane algorithm and computational results. Preprint, 1992.

[GMW92b] M. Grötschel, A. Martin, and R. Weismantel. Packing steiner trees: Polyhedral investigations. Preprint, 1992.

[Goe91] Michel X. Goemans. The steiner tree polytop and related polyhedra. Technical report, MIT, Department of Mathematics, Cambridge, MA 02139, 1991.

[Hak71] S. L. Hakimi. Steiner problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[HO92] Jianxiu Hao and James B. Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the third annual ACM-Siam Symposium on Discrete Algorithms*, pages 165–174, Orlando, Florida, 1992.

[HP91] Karla L. Hoffman and Manfred Padberg. Improved lp-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing*, 3(2):121–134, 1991.

[HR92] F. K. Hwang and Dana S. Richards. Steiner tree problems. *Networks*, 22:55–89, 1992.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In J.W. Thatcher, editor, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[Liu90] Weiguo Liu. A lower bound for the steiner tree problem in directed graphs. *Networks*, 20:765–778, 1990.

[Luc93] Abilio Lucena. Tight bounds for the steiner problem in graphs. Preprint, IRC for Process System Engineering, Imperial College, London, 1993.

[Mar92] Alexander Martin. *Packen von Steinerbäumen: Polyedrische Studien und Anwendungen*. PhD thesis, Technische Universität Berlin, 1992.

[Mur81] Bruce A. Murtagh. *Advanced Linear Programming: Computation and Practice*. McGraw-Hill, 1981.

[NOI94] Hiroshi Nagamochi, Tadashi Ono, and Toshihide Ibaraki. Implementing an efficient minimum capacity cut algorithm. Technical Report 94011, Kyoto University, 1994.

[Pap82] Christos H. Papadimitriou. *Combinatorial Optimization*. Prentice-Hall, 1982.

[PR91] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[PS89] Hans Jürgen Prömel and Angelika Steger. Das Steinerproblem in Graphen. Technical report, Forschungsinstitut für Diskrete Mathematik, Universität Bonn, 1989.

[Ray83] V. J. Rayward-Smith. The computation of nearly minimal steiner trees in graphs. *Int. J. Math. Educ. Sci. Technol.*, 14:15–23, 1983.

[RC86] V. J. Rayward-Smith and A. Clare. On finding steiner vertices. *Networks*, 16:283–294, 1986.

[SC73] J. Soukup and W. F. Chow. Set of test problems for the minimum length connection networks. *ACM/SIGMAP Newsletters*, (15):48–51, 1973.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[SDK83] Maciej M. Syslo, Narsingh Deo, and Janusz S. Kowalik. *Discrete Optimization Algorithms.* Prentice-Hall, 1983.

[TM80] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Math. Jpon.*, 24:573–577, 1980.

[WI88] B. M. Waxman and M. Imase. Worst-case performance of the rayward-smith's steiner tree heuristics. *Inform. Process. Lett.*, 29:283–287, 1988.

[Win87] Pawel Winter. Steiner problems in networks: A survey. *Networks*, 17:129–167, 1987.

[Won84] Richard T. Wong. A dual ascent approach for the steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.

[WS92] Pawel Winter and J. MacGregor Smith. Path-distance heuristics for the steiner problem in undirected networks. *Algorithmica*, pages 309–327, 1992.