Bikriterielle Optimierung auf Bäumen

GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Bachelorarbeit in Mathematik
eingereicht an der Fakultät für Mathematik und Informatik
der Georg-August-Universität Göttingen
am 09. Februar 2012

von

Stefan Schäfer

Erstgutachterin:

Prof. Dr. Anita Schöbel

Zweitgutachter:

Jun.-Prof. Dr. Stephan Westphal

Inhaltsverzeichnis

1 Einleitung

Die Optimierung ist heutzutage Bestandteil in nahezu jedem Bereich der Wirtschaft, Technik und Forschung. In dieser Arbeit beschäftigen wir uns im Speziellen mit der Optimierung in der Graphentheorie. Ob Routenberechnung, Netzwerktheorie oder Bildbearbeitung – Anwendungsbeispiele gibt es zahlreiche.
Graphen bestehen aus einer Knoten- und einer Kantenmenge, wobei die Kanten Verbindungen oder Beziehungen zwischen den Knoten darstellen. Anschauliche Beispiele sind Straßennetze oder Stammbäume. Im Folgenden behandeln wir Graphen, bei denen alle Knoten miteinander verbunden sind, ohne dass Kreise entstehen. Diese werden spannende Bäume genannt und beispielsweise in der Informatik für Datenstrukturen, oder in der Technik zur Planung von Telefonleitungen oder Bewässerungssystemen verwendet.
Wie auch bei realen Anwendungen hat man bestimmte Kosten oder Gewichte auf den Kanten, zum Beispiel Entfernungen, Zeiten oder Gebühren. Diese Kosten möchte man minimieren und erhält als Lösung minimal spannende Bäume, welche zur Modellierung kostengünstigster Grundgerüste vieler Probleme dienen. Ein bekanntes Beispiel zum Lösen dieses sogenannten Minimum Spanning Tree Problems ist Prim's Algorithmus.
In vielen Fällen spielen mehrere verschiedene Kriterien eine Rolle, die sich unter Umständen gegenseitig beeinflussen oder gar miteinander in Konflikt stehen. Beispielsweise erreicht eine Spedition durch Befahren von Autobahnen statt Landstraßen die Ziele schneller, zugleich entstehen aber höhere Kosten durch Mautgebühren oder höheren Spritverbrauch. Hier findet die multikriterielle Optimierung Anwendung, bei welcher die einzelnen Kostenkomponenten minimiert werden sollen, ohne andere zu verschlechtern. Die entstehenden Lösungen werden effizient oder Pareto optimal genannt.
Mit Hilfe dieser Grundlagen betrachten wir das Minimum Spanning Tree Problem im Multikriteriellen. Wir untersuchen in Kapitel 3 notwendige Bedingungen von effizienten Bäumen und analysieren die von Hamacher und Ruhe [14] vorgestellte mehrkriterielle Version von Prim's Algorithmus. Wir beweisen, dass diese nicht alle effizienten Bäume findet und geben eine korrigierte Variante an. Der entstehende Algorithmus ist allerdings aufgrund einer sehr hohen Laufzeit nicht praxistauglich, weshalb Alternativen zur Berechnung aller minimal spannenden Bäume wünschenswert wären.

Eine Möglichkeit bietet eine einfache Nachbarschaftssuche (engl.: Neighborhood Search), bei welcher der Austausch von Kanten zu neuen optimalen Bäumen führt. Hierzu gibt es bereits viele Algorithmen, beispielsweise von Andersen u. a. [1] oder Shioura u. a. [20], die das Problem effizient lösen würden, vorausgesetzt, dass die Lösungen in einer spezifischen Art und Weise zusammenhängen. Dies werden wir in Kapitel 4 graphentheoretisch anhand des Pareto-Graphen beschreiben und damit den Zusammenhang der effizienten Menge definieren.
Wie Ehrgott und Klamroth [9] zeigten, ist die Menge der minimal spannenden Bäume im Allgemeinen allerdings nicht zusammenhängend und jeder Graph kann erweitert werden, so dass ein Neighborhood Search Algorithmus nicht alle Lösungen finden kann. Dies motiviert die Suche nach Graphen unter bestimmten Restriktionen, deren zugehörige effiziente Menge zusammenhängend ist. Dabei betrachten wir insbesondere die Einschränkung auf bikriterielle Kosten. Gorski [11] bewies, dass beim zweikriteriellen Minimum Spanning Tree Problem mit binären Kosten in der zweiten Komponente die zugehörige Lösungsmenge immer zusammenhängend ist.
Ausgehend von diesen Erkenntnissen betrachten wir in Kapitel 5 Graphen mit bikriteriellen Kosten, deren Kosten in der ersten Komponente beliebig, in der zweiten allerdings nur Elemente aus $\{0, 1, 2\}$ sein dürfen. Es ist ein offenes Problem, ob durch diese Erweiterung der binären Kosten um den zusätzlichen Wert 2 die effiziente Menge zusammenhängend bleibt.
Dieses Problem wollen wir numerisch lösen, weshalb wir Algorithmen zum Erstellen von zufälligen Testgraphen, die korrigierte multikriterielle Version von Prim's Algorithmus und Verfahren zur Überprüfung des Zusammenhangs der effizienten Menge betrachten und in C++ programmieren. In Abhängigkeit von der Knoten- und Kantenanzahl werden wir zufällige Graphen und deren Pareto-Graphen erzeugen, zeichnen und auf Zusammenhang überprüfen.
Abschließend folgt in Kapitel 6 eine kurze Zusammenfassung der vorgestellten Erkenntnisse und Experimente, sowie ein Ausblick bezüglich weiterer Forschungen.

2 Grundbegriffe

Um überhaupt mit den in der Einleitung beschriebenen Problemen arbeiten zu können, werden wir im Folgenden zuerst die benötigten Grundbegriffe, Sätze und Definitionen betrachten. Wir beginnen mit der Graphentheorie und dem Minimum Spanning Tree Problem im Einkriteriellen. Anschließend werden wir die Grundlagen der multikriteriellen Optimierung einführen, um diese im nächsten Kapitel auf das Minimum Spanning Tree Problem übertragen zu können.

2.1 Graphentheorie

Graphen werden durch eine Knoten- und eine Kantenmenge definiert, wobei die Kanten die Verbindungen zwischen den Knoten darstellen. Die Knoten repräsentieren zum Beispiel Orte, Gegenstände oder Ereignisse, welche in irgendeiner Art verbunden sind oder zusammenhängen. Zum Beispiel Städte, die verbunden sind durch Straßen und Wege oder Ankunfts- und Abfahrtsereignisse an Bahnhöfen, zwischen denen Züge fahren oder Passagiere umsteigen.
Die folgenden Grundlagen können in jedem Standardwerk über Graphentheorie nachgelesen werden. Allerdings variieren die Notationen häufig. Die folgenden Definitionen und Sätze orientieren sich an den Büchern von Hamacher und Klamroth [13], Korte und Vygen [16], sowie Krumke und Noltemeier [17], und eigenen Mitschriften der Vorlesung *Netzwerkflüsse* von Westphal [22].

Definition 2.1 (Grundbegriffe)**.**

1. Ein *ungerichteter Graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ist ein Tupel bestehend aus
   - einer nichtleeren, endlichen Menge $\mathcal{V} = \mathcal{V}(\mathcal{G})$ von *Knoten*
   - einer Menge $\mathcal{E} = \mathcal{E}(\mathcal{G}) \subseteq \mathcal{V} \times \mathcal{V}$ von *Kanten*
2. Jede Kante $e \in \mathcal{E}$ hat zwei *Endpunkte* u, $v \in \gamma(e) \subseteq \mathcal{V}$. Dabei gilt:
   - u und v sind *inzident* zu e,
   - u und v sind *adjazent* zueinander.

   Wir schreiben $e = [u, v] = [v, u]$.
3. Ein Knoten heißt *isoliert*, wenn er zu keiner Kante inzident ist.

4. Die Kante e ist eine *Schleife*, falls $u = v$, und zwei Kanten e_1, e_2 heißen *parallel*, falls $\gamma(e_1) = \gamma(e_2)$, wenn sie also die gleichen Endpunkte haben.

5. Der Graph $\mathcal{G}$ heißt *einfach*, falls er weder Schleifen noch parallele Kanten enthält.

Bemerkung. $\mathcal{E}$ ist eine symmetrische Relation auf der Knotenmenge $\mathcal{V}$.

Nachfolgend sei ein Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit $|\mathcal{V}| = n$ und $|\mathcal{E}| = m$ gegeben. Wir möchten nun die Verbindungen von Knoten näher beschreiben, dafür betrachten wir folgende Definition.

Definition 2.2 (Wege und Zusammenhang)**.**

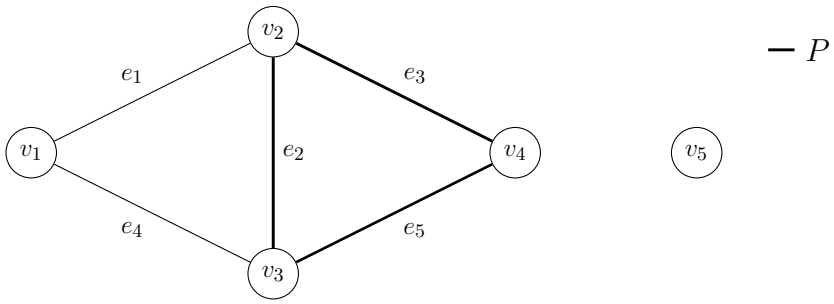
1. Ein *Weg* oder *Pfad* P in $\mathcal{G}$ ist eine endliche Folge $(v_0, e_1, v_1, e_2, \ldots, e_k, v_k)$ mit $v_0, \ldots, v_k \in \mathcal{V}$, $e_1, \ldots, e_k \in \mathcal{E}$ und $e_i = [v_{i-1}, v_i]$ für $i = 1, \ldots, k$. Die *Länge* des Weges ist k.

2. Ein Weg P heißt *einfach*, falls er keine Kante wiederholt und *Kreis*, falls $v_0 = v_k$ gilt.

3. Der Graph $\mathcal{G}$ heißt *zusammenhängend*, falls zu je zwei Knoten u, $v \in \mathcal{V}$ ein einfacher Weg existiert, der sie verbindet und *azyklisch*, falls er keinen Kreis enthält.

Bemerkung. In manchen Fällen beschreibt man einen Weg auch nur durch die vorkommenden Knoten oder Kanten.

Beispiel 2.3. Hier haben wir einen einfachen Graphen $\mathcal{G}$ mit fünf Knoten $v_1, \ldots, v_5$ und fünf Kanten $e_1, \ldots, e_5$ gegeben, der nicht zusammenhängend ist, da der Knoten v_5 isoliert ist.



Damit existieren Wege zwischen den Knoten v_1 bis v_4, aber nicht zu v_5. Der Weg $P = (v_2, e_2, v_3, e_5, v_4, e_3, v_2)$ ist ein einfacher Kreis der Länge 3.

Wir werden gegebene Graphen nicht ausschließlich komplett betrachten, sondern häufig nur Teile von Graphen. Im Speziellen werden wir uns in dieser Arbeit auf Bäume konzentrieren, welche hier zuerst definiert werden.

Definition 2.4 (Teilgraphen und Bäume)**.**

1. Ein Graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ heißt *Teilgraph* von $\mathcal{G}$, falls $\mathcal{V}' \subseteq \mathcal{V}$ und $\mathcal{E}' \subseteq \mathcal{E}$ mit $\mathcal{E}' \subseteq \mathcal{V}' \times \mathcal{V}'$.

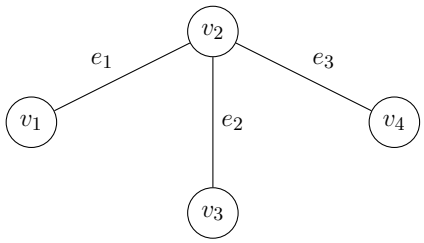
2. Maximal zusammenhängende Teilgraphen heißen *Zusammenhangskomponenten.*

3. Ein Teilgraph $\mathcal{H}$ von $\mathcal{G}$ heißt *spannend*, falls $\mathcal{V}(\mathcal{H}) = \mathcal{V}(\mathcal{G})$ gilt.

4. Ein Graph $\mathcal{G}$ heißt *Baum*, falls $\mathcal{G}$ kreisfrei und zusammenhängend ist. Damit ist ein *spannender Baum* T von $\mathcal{G}$ ein Baum, der spannender Teilgraph von $\mathcal{G}$ ist.

Bemerkung. Ein spannender Baum eines Graphen $\mathcal{G}$ existiert genau dann, wenn $\mathcal{G}$ zusammenhängend ist.

Beispiel 2.5. Der folgende Graph ist ein Teilgraph von $\mathcal{G}$ aus Beispiel 2.3, welcher zugleich auch ein Baum ist:



Wir betrachten später Teilmengen $X \subset \mathcal{V}$ und dazu die Menge von Kanten, die die Knoten aus X mit denen aus $\mathcal{G}\backslash X$ verbinden. Dafür definieren wir den Schnitt einer Knotenmenge.

Definition 2.6 (Schnitt)**.**
Die Menge der zu v inzidenten Kanten wird mit $\delta(v)$ bezeichnet und für eine nichtleere Knotenmenge $X \subset \mathcal{V}$ ist der *Schnitt von* X definiert als

$$\delta(X) \coloneqq \{e \in E \,|\, e \text{ hat einen Endpunkt in X und einen in } \mathcal{V}\backslash X\}.$$

Der Schnitt einer Knotenmenge X ist demnach die Menge von Kanten, die aus X heraus- beziehungsweise hineinzeigen.
Mit den eingeführten Grundbegriffen und Definitionen sind wir in der Lage, unsere ersten Aussagen über Bäume zu treffen. Für den Beweis des folgenden Lemmas und des Satzes siehe Korte und Vygen [16, Prop. 2.3 a) und Theor. 2.4].

Lemma 2.7. *Ein ungerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ist genau dann zusammenhängend, wenn der Schnitt $\delta(X)$ nichtleer ist, für alle Mengen $\emptyset \neq X \subset \mathcal{V}$.*

Satz 2.8 (Äquivalente Aussagen für Bäume)**.**
Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein ungerichteter Graph mit $|\mathcal{V}| = n$. Dann sind äquivalent:

a) *$\mathcal{G}$ ist Baum*

b) *$\mathcal{G}$ hat $n-1$ Kanten und keinen Kreis*

c) *$\mathcal{G}$ hat $n-1$ Kanten und ist zusammenhängend*

d) *$\mathcal{G}$ ist minimal zusammenhängend*

e) *$\mathcal{G}$ ist ein minimaler Graph mit $\delta(X) \neq \emptyset$, für alle $\emptyset \neq X \subsetneq \mathcal{V}$*

f) *$\mathcal{G}$ ist maximal kreisfrei*

g) *Jedes Knotenpaar in $\mathcal{G}$ ist durch einen eindeutigen Weg verbunden*

Der Vollständigkeit halber sind hier mehr Punkte aufgezählt als wir später brauchen.

2.2 Minimum Spanning Tree Problem

Im Folgenden betrachten wir Kosten beziehungsweise Gewichte auf den Kanten. Im Allgemeinen sind diese gegeben durch eine reellwertige Funktion auf den Kanten

$$c\colon \mathcal{E} \to \mathbb{R},$$

wobei wir später nur ganzzahlige nichtnegative Kosten betrachten werden.
Beim Minimum Spanning Tree Problem (MSTP) versucht man eine möglichst günstige Verbindung aller Knoten eines Graphen zu finden. Dies kann man sich als ein minimales Grundgerüst vorstellen, wie zum Beispiel ein kostengünstigstes

Stromnetz, welches die Grundversorgung aller Haushalte sichert, oder ein kleinstes Computernetzwerk, das die Verbindung aller Rechner untereinander garantiert. Die entstehenden Graphen werden minimal spannende Bäume (MST) genannt. Nachfolgend seien alle betrachteten Graphen zusammenhängend, da nach Definition 2.4 spannende Bäume nur in zusammenhängenden Graphen existieren. Damit ist das Problem gegeben durch:

The Minimum Spanning Tree Problem

Instanz: Einfacher, zusammenhängender, ungerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Kantengewichte $c\colon \mathcal{E} \to \mathbb{R}$

Aufgabe: Finde spannenden Baum T in $\mathcal{G}$ mit minimalem Gewicht

$$c(T) \coloneqq \sum_{e \in \mathcal{E}(T)} c(e).$$

Es gibt äquivalente Aussagen über minimal spannende Bäume, mit deren Hilfe man das MSTP lösen kann. Diese Eigenschaften liefert uns der folgende Satz, für den Beweis siehe zum Beispiel Korte und Vygen [16, Theor. 6.2].

Satz 2.9 (Optimalitätskriterien für MST)**.**
Seien $(\mathcal{G}, c)$ eine Instanz des Minimum Spanning Tree Problems und T ein spannender Baum in $\mathcal{G}$, dann sind äquivalent:

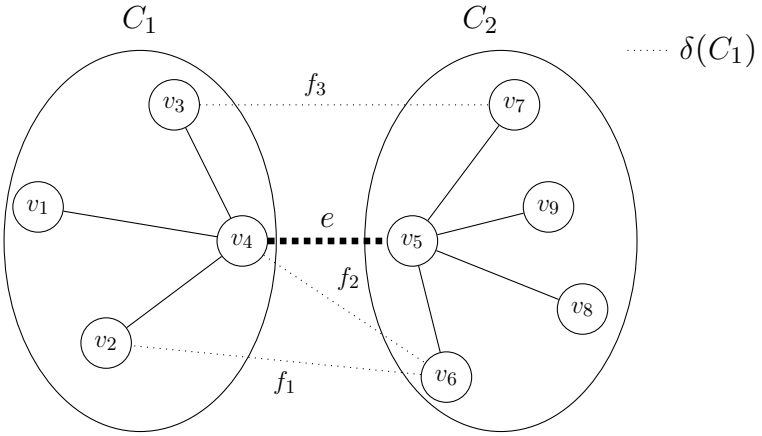
a) *T ist optimal*

b) *Für $e \in \mathcal{E}(T)$ ist $c(e) \in \min\{c(f) \mid f \in \delta(C)\}$,*
C Zusammenhangskomponente von $T - e$.

c) *Sei $f = (x, y) \in \mathcal{E}(\mathcal{G}) \backslash \mathcal{E}(T)$ und $P(f)$ der eindeutige x-y-Pfad in T.*
Dann ist $c(f) < c(e)$ für kein $e \in P(f)$ erfüllt.

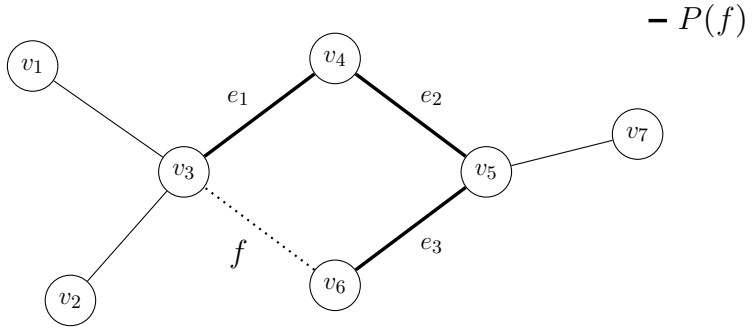
Punkt a) bedeutet, dass T ein minimal spannender Baum ist. Zur Verdeutlichung des Satzes betrachten wir zwei Beispiele zu den beiden äquivalenten Aussagen b) und c).

Beispiel 2.10. Wir betrachten folgenden minimal spannenden Baum, welcher durch e die beiden Zusammenhangskomponenten C_1 und C_2 verbindet.



Aus Satz 2.9 b) folgt, dass im Schnitt $\delta(C_1)$ die Kosten von e minimal sind, also f_1, f_2 und f_3 nicht kleinere Kosten haben können. Das heißt, dass ein Baum genau dann optimal ist, wenn er minimale Kanten aus Schnitten seiner Zusammenhangskomponenten enthält.

Beispiel 2.11. Sei folgender minimal spannender Baum mit einer zusätzlichen Kante f aus dem ursprünglichen Graphen gegeben.



Nach Punkt c) aus dem vorherigen Satz 2.9 sind die Kosten von e_1, e_2, e_3 kleiner oder gleich $c(f)$. Damit ist ein Baum genau dann optimal, wenn er minimale Kanten aus Kreisen des ursprünglichen Graphen enthält.

Aufbauend auf Kriterium b) hat Prim [19] folgenden Algorithmus zum Lösen des Minimum Spanning Tree Problems vorgestellt:

Algorithm 1: Prim's Spanning Tree Algorithmus

Input : Einfacher, zusammenhängender, ungerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Kantengewichte $c\colon \mathcal{E} \to \mathbb{Z}$

Output : Minimal spannender Baum T

1 Wähle $v_0 \in \mathcal{V}(\mathcal{G})$ beliebig, setze $T \coloneqq (\{v_0\}, \emptyset)$
2 **while** $\mathcal{V}(T) \neq \mathcal{V}(\mathcal{G})$ **do**
3 Wähle $e \in \delta_{\mathcal{G}}(\mathcal{V}(T))$ mit minimalen Kosten, $T = T + e$
4 **end**
5 **return** MST T

Der Algorithmus startet mit einem beliebigen Knoten und fügt in jedem Schritt eine Kante mit minimalen Kosten aus dem Schnitt des bisherigen Teilgraphen hinzu, so lange nicht alle Knoten verbunden sind.
Dabei ist $T = T + e$ in Schritt 3 eine Kurzschreibweise für $\mathcal{E}(T) = \mathcal{E}(T) \cup \{e\}$ und $\mathcal{V}(T) = \mathcal{V}(T) \cup \gamma(e)$, das heißt, dass der entsprechende Endpunkt der Kante e auch hinzugefügt wird.

Satz 2.12. *Der Algorithmus von Prim ist korrekt und lässt sich in $\mathcal{O}(n^2)$ Schritten implementieren. Durch Verwendung von Fibonacci-Heaps beträgt die Laufzeit $\mathcal{O}(m + n \log n)$.*

Beweis. Nach jeder Iteration der while-Schleife erfüllt T die Bedingung b) aus Satz 2.9 im betrachteten Teilgraphen, somit auch nach der letzen Iteration. Damit ist T nach Satz 2.9 ein minimal spannender Baum.
Für den Beweis der Laufzeit siehe zum Beispiel Korte und Vygen [16, Theor. 6.5 und Kor. 6.7]. □

Bemerkung. Es gibt noch weitere Algorithmen zum Bestimmen von minimal spannenden Bäumen. Sehr bekannt ist zum Beispiel der Algorithmus von Kruskal [18]. Dieser und Prim's Algorithmus gehen zurück auf die Ideen von Jarnik [15].

Bemerkung. Die Laufzeit von Algorithmen für das MSTP wurde sehr intensiv untersucht und mehrmals verbessert, beispielsweise von Gabow u. a. [10] auf $\mathcal{O}(m \log \beta(m, n))$ mit $\beta(m, n)$ gleich der Anzahl notwendiger log-Iterationen, um n auf eine Zahl kleiner gleich m/n abzubilden, oder von Chazelle [4] auf $\mathcal{O}(m\alpha(m, n))$, wobei α die Inverse der Ackermann-Funktion ist.

2.3 Multikriterielle Optimierung

Die folgenden Grundlagen der multikriteriellen Optimierung richten sich nach dem Buch von Ehrgott [8].
In einem Teilgebiet der Optimierung behandelt man Problemstellungen mit mehreren Zielfunktionen. Es werden beispielsweise gleichzeitig mehrere verschiedene Kosten minimiert, die sich unter Umständen aber gegenseitig beeinflussen oder gar miteinander in Konflikt stehen. Dadurch wird das Erreichen aller Ziele sehr schwierig oder sogar unmöglich.
Mit diesen Problemen beschäftigt man sich in der multikriteriellen Optimierung. Wir betrachten multikriterielle Optimierungsprobleme (MOP) auf dem $\mathbb{R}^p$ der Form

$$\min_{x\in\mathcal{X}} f(x) = \min_{x\in\mathcal{X}}(f_1(x),\dots,f_p(x)).$$

Dabei ist $\mathcal{X}$ die *zulässige Menge*, die durch den *Zielfunktionsvektor*

$$f = (f_1,\dots,f_p)\colon \mathcal{X} \to \mathbb{R}^p$$

in den *Werteraum* $\mathbb{R}^p$ abgebildet wird. Das Bild der zulässigen Menge $\mathcal{X}$ unter f bezeichnen wir mit

$$\mathcal{Y} \coloneqq f(\mathcal{X}) \coloneqq \{y \in \mathbb{R}^p \mid y = f(x),\, x \in \mathcal{X}\}.$$

Es ist zunächst nicht klar, was $\min_{x\in\mathcal{X}}$ im Multikriteriellen heißt. Wir betrachten das Konzept der *Pareto-Effizienz*, welches vorsieht, dass eine effiziente Lösung in keinem Parameter verbessert werden kann, ohne andere Parameter zu verschlechtern. Dieses Konzept basiert auf folgender Relation im $\mathbb{R}^p$:

Definition 2.13.
Für zwei Vektoren $y^1, y^2 \in \mathbb{R}^p$ gilt

$$y^1 \leq y^2 \quad :\Leftrightarrow \quad y^1_k \leq y^2_k, \quad k = 1,\dots,p \quad \text{und} \quad y^1 \neq y^2.$$

Wenn $y^1 \leq y^2$ gilt, ist demnach y^1 in jeder Komponente kleiner oder gleich y^2 und die Ungleichung ist in mindestens einer Komponente strikt.

Bemerkung. Die Relation "$\leq$" ist asymmetrisch und transitiv und damit eine strikte Halbordnung.

Durch diese Wahl der Relation klären wir die Bedeutung von $\min_{x \in \mathcal{X}}$ im Multikriteriellen. Dazu betrachten wir folgende Definition:

Definition 2.14 (Effizienz und Nichtdominiertheit)**.**
Eine zulässige Lösung $x \in \mathcal{X}$ heißt *effizient* oder *Pareto optimal*, falls kein $x' \in \mathcal{X}$ existiert, so dass $f(x') \leq f(x)$ gilt. Für ein effizientes x heißt $f(x)$ *nichtdominierter Punkt*.
Falls $x^1, x^2 \in \mathcal{X}$ und $f(x^1) \leq f(x^2)$ gelten, sagen wir x^1 *dominiert* x^2 und $f(x^1)$ *dominiert* $f(x^2)$.
Die *effiziente Menge* $\mathcal{X}_E$ aller effizienten Lösungen ist definiert als

$$\mathcal{X}_E \coloneqq \{x \in \mathcal{X} \mid \text{Es existiert kein } x' \in \mathcal{X} \text{ mit } f(x') \leq f(x)\}.$$

Das Bild der effizienten Menge unter f wird mit $\mathcal{Y}_N \coloneqq f(\mathcal{X}_E)$ bezeichnet und heißt *nichtdominierte Menge*.

Anders als im Einkriteriellen erhalten wir nach Definition 2.14 meist keine eindeutige Lösung mit minimalem Zielfunktionswert, sondern lösen multikriterielle Optimierungsprobleme durch Berechnen der effizienten Menge.
Die erhaltenen effizienten Lösungen haben alle nichtvergleichbare, nichtdominierte Kostenvektoren und können somit als "optimal" interpretiert werden. Denn eine Lösung eines Problems dominiert eine andere nur, falls sie in allen Komponenten kleiner gleich und in einer Komponente echt besser ist.
Beispielsweise kann ein Aufgabensteller dadurch verschiedene Möglichkeiten abwägen: Möchte er eine bestimmte Komponente stärker berücksichtigen und damit höhere Kosten der anderen Zielfunktionen in Kauf nehmen, oder sind ungefähr gleich hohe Kosten in allen Komponenten erwünscht?

3 Multikriterielle spannende Bäume

Die Grundlagen aus Kapitel 2 übertragen wir im Folgenden auf die multikriterielle Optimierung auf Bäumen. Wir betrachten zusammenhängende ungerichtete Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit multikriteriellen Kantengewichten, das heißt die Kostenfunktion

$$c\colon \mathcal{E} \to \mathbb{Z}^p$$

besteht aus p ganzzahligen Komponentenfunktionen

$$c^k\colon \mathcal{E} \to \mathbb{Z}, \quad k = 1, \ldots, p.$$

Als zulässige Menge $\mathcal{X}$ betrachten wir die Menge aller spannenden Bäume in $\mathcal{G}$

$$\mathcal{T} \coloneqq \{T \mid T \text{ ist spannender Baum in } \mathcal{G}\}.$$

3.1 Multikriterielles Minimum Spanning Tree Problem

Im Speziellen betrachten wir das Minimum Spanning Tree Problem im Multikriteriellen. Wir bestimmen die spannenden Bäume mit nichtdominierten Kosten bezüglich der komponentenweisen strikten Halbordnung "$\leq$" aus Definition 2.13. Im Gegensatz zum minimal spannenden Baum mit eindeutigem Gewicht im Einkriteriellen aus Kapitel 2.2 erhalten wir eine Menge von spannenden Bäumen, die alle nichtvergleichbare, nichtdominierte Kosten haben.
Die so erhaltenen minimal spannenden Bäume im Multikriteriellen werden auch *effiziente spannende Bäume* (EST) genannt.
Damit ist das multikriterielle Minimum Spanning Tree Problem gegeben durch:

Multikriterielles Minimum Spanning Tree Problem

Instanz: Einfacher, zusammenhängender, ungerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Kantengewichte $c\colon \mathcal{E} \to \mathbb{Z}^p$

Aufgabe: Finde alle spannenden Bäume in $\mathcal{G}$ mit minimalem Gewicht:

$$\min_{T \in \mathcal{T}} c(T) \coloneqq \min_{T \in \mathcal{T}} \sum_{e \in \mathcal{E}(T)} c(e).$$

Notation. Wir schreiben $\mathcal{T}_E$ anstelle von $\mathcal{X}_E$ für die *effiziente Menge* des Minimum Spanning Tree Problem im Multikriteriellen.

Effiziente spannende Bäume haben ähnliche Eigenschaften wie minimal spannende Bäume im Einkriteriellen. In Satz 2.9 haben wir äquivalente Aussagen zur Optimalität aufgeführt. Für effiziente spannende Bäume erhalten wir allerdings nur zwei notwendige Kriterien, welche Hamacher und Ruhe [14] veröffentlichten.

Satz 3.1. *Sei T ein effizienter spannender Baum von $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Dann gilt:*

1) *Für $e \in \mathcal{E}(T)$ ist $c(e) \in \min\{c(f) \mid f \in \delta(C)\}$,*
   C Zusammenhangskomponente von $T - e$.

2) *Sei $f = (x, y) \in \mathcal{E}(\mathcal{G}) \backslash \mathcal{E}(T)$ und $P(f)$ der eindeutige x-y-Pfad in T.*
   Dann ist $c(f) \leq c(e)$ für kein $e \in P(f)$ erfüllt.

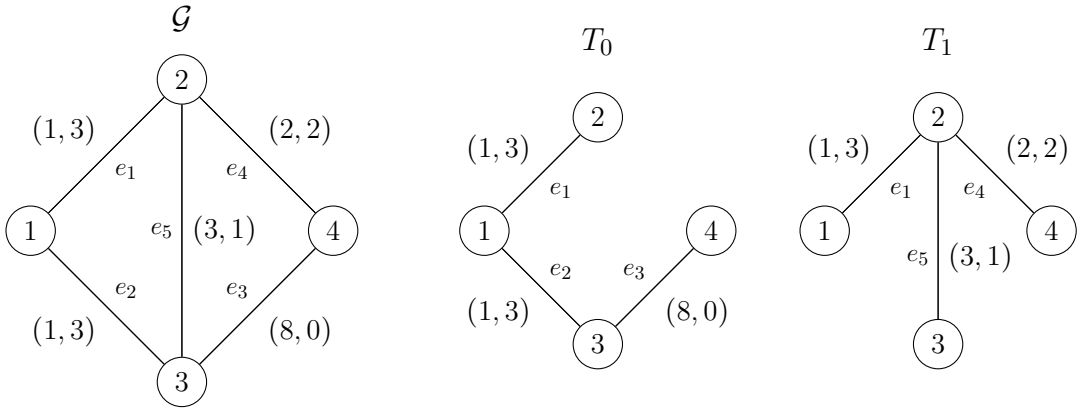
Beweis. Wir beweisen beide Punkte zusammen. Sei T ein effizienter spannender Baum von $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Angenommen 1) und 2) seien nicht erfüllt.
Dann definieren wir einen neuen spannenden Baum $T' = (\mathcal{V}, \mathcal{E}(T'))$ mit

$$\mathcal{E}(T') = (\mathcal{E}(T) \backslash \{e\}) \cup \{f\},$$

für ein $f \in \delta(C)$ beziehungsweise $e \in P(f)$ mit $c(f) \leq c(e)$. Damit wäre allerdings $c(T') = c(T) - c(e) + c(f) \leq c(T)$. Dies ist ein Widerspruch zur Effizienz von T. □

Die Kriterien sind analog zu den Bedingungen b) und c) aus Satz 2.9 formuliert, wobei hier das Minimum bei 1) und "$\leq$" in 2) entsprechend der komponentenweisen strikten Halbordnung aus Definition 2.13 definiert sind. Zur Veranschaulichung betrachte die Beispiele 2.10 und 2.11.
Der Satz 3.1 geht auf Corley [6] zurück, der allerdings fälschlicherweise von der Äquivalenz der Aussagen im Satz ausging. Hamacher und Ruhe [14] zeigten, dass die Bedinungen nur notwendig sind. Dazu betrachten wir folgendes Beispiel, welches eine korrigierte Version von Ehrgott [8] ist.

Beispiel 3.2. Wir betrachten folgenden Graphen $\mathcal{G}$ und zwei zugehörige spannende Bäume T_0 und T_1.



$c(T_0) = (10, 6)$ $\qquad$ $c(T_1) = (6, 6)$

Der Baum T_0 wird von T_1 dominiert. Wir überprüfen, ob T_0 trotzdem die beiden Eigenschaften aus Satz 3.1 erfüllt. Für Kriterium 1) muss T_0 minimale Kanten in Schnitten haben. Für den Schnitt von $T_0 - e_1$ gilt

$$\delta(T_0 - e_1) = \delta(2) = \{e_1, e_4, e_5\}.$$

Die Kante e_1 erfüllt die erste Bedingung, denn es gilt:

$$c(e_1) = (1, 3) \in \min\{(1, 3), (2, 2), (3, 1)\} = \min\{c(e_1), c(e_4), c(e_5)\}.$$

Analog gilt für e_2 und e_3:

$$\delta(T_0 - e_2) = \{e_2, e_4, e_5\} \text{ und } c(e_2) \in \min\{c(e_2), c(e_4), c(e_5)\},$$

$$\delta(T_0 - e_3) = \{e_3, e_4\} \text{ und } c(e_3) \in \min\{c(e_3), c(e_4)\}.$$

Somit ist das erste Kriterium für T_0 erfüllt. Wir überprüfen Kriterium 2), welches besagt, dass effiziente spannende Bäume minimale Kanten aus Kreisen haben. Betrachte zuerst $e_5 = [2, 3]$ und den zugehörigen Pfad $P(e_5) = (e_1, e_2)$. Wir erkennen, dass

$$c(e_5) = (3, 1) \not\leq (1, 3) = c(e_1) = c(e_2).$$

Für $e_4 = [2, 4]$ erhalten wir $P(e_4) = (e_1, e_2, e_3)$ und auch hier gilt:

$$c(e_4) = (2, 2) \not\leq (1, 3) = c(e_1) = c(e_2),$$

$$c(e_4) = (2, 2) \not\leq (8, 0) = c(e_3).$$

Somit sind beide Kriterien aus Satz 3.1 erfüllt, obwohl T_0 nicht effizient ist. Demnach sind die Kriterien für die Optimalität eines Baumes nur notwendig, aber nicht hinreichend.

3.2 Multikriterieller Algorithmus von Prim

Aufbauend auf dem Algorithmus von Corley [6] stellten Hamacher und Ruhe [14] eine multikriterielle Version für Prim's Algorithmus vor. Der Algorithmus findet sich beispielsweise auch bei Ehrgott [8].

Algorithm 2: Prim's Spanning Tree Algorithmus - Multikriteriell

Input : Einfacher, zusammenhängender, ungerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Kantengewichte $c\colon \mathcal{E} \to \mathbb{Z}^p$

Output: $\mathcal{T}_{n-1}$, die Menge aller MST von $\mathcal{G}$

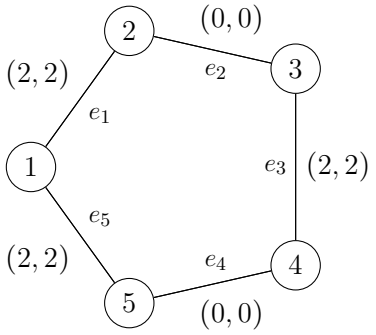
1 Bestimme $\mathcal{T}_1 \coloneqq argmin\{c(e_1), \ldots, c(e_m)\}$
2 **for** $k = 2, \ldots, n-1$ **do**
3 Berechne $\mathcal{T}_k \coloneqq \{\, \mathcal{E}(T) \cup \{f\} \,|\, T \in \mathcal{T}_{k-1}\,,\ f \in argmin\{c(e) \,|\, e \in \delta(T)\} \,\}$
4 Setze $\mathcal{T}_k \coloneqq argmin\{c(T) \,|\, T \in \mathcal{T}_k\}$
5 **end**
6 **return** $\mathcal{T}_{n-1}$ die Menge aller effizienten spannenden Bäume von $\mathcal{G}$

Der Algorithmus bestimmt am Anfang die Kanten mit nichtdominierten Kosten. In den nachfolgenden $n-2$ Iterationen wird für jeden bisherigen Teilbaum jeweils ein neuer Teilbaum durch Hinzufügen einer minimalen Kante aus dem Schnitt erzeugt. Dabei wird in Schritt 3 eine Menge von Teilbäumen gebildet, in der möglicherweise doppelt erzeugte Teilbäume aussortiert werden, da jedes Element in einer Menge nur einmal vorkommt.
Die so erhaltenen Graphen sind nach Lemma 2.7 zusammenhängend und haben $n-1$ Kanten und sind daher nach Satz 2.8 c) spannende Bäume.
Sie sind effizient, da sie die notwendige Bedingung 1) aus Satz 3.1 erfüllen und

zusätzlich alle dominierten Teilbäume in Schritt 4 aussortiert werden.
Jedoch findet der Algorithmus nicht alle Lösungen des Problems, wie der folgende Satz zeigt.

Satz 3.3. *Der vorgestellte Algorithmus 2, als multikriterielle Version von Prim's Algorithmus, findet nicht alle effizienten spannenden Bäume.*

Beweis. Wir betrachten folgenden Graphen $\mathcal{G}$ als Gegenbeispiel:



Es existieren genau drei minimal spannende Bäume T_1, T_2 und T_3 mit

$$\mathcal{E}(T_1) = \{e_1, e_2, e_3, e_4\},$$
$$\mathcal{E}(T_2) = \{e_2, e_3, e_4, e_5\},$$
$$\mathcal{E}(T_3) = \{e_1, e_2, e_4, e_5\}.$$

Jeder davon enthält die beiden Kanten e_2 und e_4 mit Kosten $(0, 0)$ und zusätzlich zwei der drei Kanten e_1, e_3 oder e_5 mit Kosten $(2, 2)$.
Wenden wir Algorithmus 2 auf den Graphen $\mathcal{G}$ an, erhalten wir allerdings Folgendes:
In Schritt 1 bestimmen wir

$$\mathcal{T}_1 \coloneqq argmin\{c(e_1), \ldots, c(e_5)\} = \{e_2, e_4\}.$$

In der ersten Iteration der for-Schleife fügen wir zu den beiden Kanten aus $\mathcal{T}_1$ jeweils eine minimale Kante aus dem Schnitt hinzu. In diesem Fall haben alle Kanten aus den Schnitten Kosten $(2, 2)$, weshalb gilt

$$\mathcal{T}_2 = \{\{e_1, e_2\}, \{e_2, e_3\}, \{e_3, e_4\}, \{e_4, e_5\}\}.$$

Jeder Teilbaum hat Kosten $(2, 2)$. Daher löschen wir in Schritt 4 keinen der Teilgraphen aus $\mathcal{T}_2$.

Für $k = 3$ erhalten wir analog wie oben

$$\mathcal{T}_3 = \{\{e_1, e_2, e_3\}, \{e_1, e_2, e_5\}, \{e_2, e_3, e_4\}, \{e_3, e_4, e_5\}\{e_1, e_4, e_5\}\}.$$

Der Teilbaum $\{e_2, e_3, e_4\}$ hat Kosten $(2, 2)$, alle anderen haben Kosten $(4, 4)$. Damit werden die Dominierten in Schritt 4 gelöscht und wir erhalten

$$\mathcal{T}_3 = \{\{e_2, e_3, e_4\}\}.$$

Der Schnitt des Teilbaums $\{e_2, e_3, e_4\}$ enthält in der letzten Iteration e_1 und e_5, die beide Kosten $(2, 2)$ haben. Daher folgt

$$\mathcal{T}_4 = \{\{e_1, e_2, e_3, e_4\}, \{e_2, e_3, e_4, e_5\}\}.$$

Beide berechneten Bäume haben Kosten $(4, 4)$, weshalb wir in Schritt 4 keinen der beiden löschen und $\mathcal{T}_4$ zurückgeben.
Allerdings haben wir mit dem Algorithmus den minimal spannenden Baum T_3 nicht gefunden. Damit folgt die Behauptung. □

Der Algorithmus 2 sortiert in Schritt 4 dominierte Teilbäume aus. Dadurch werden allerdings, wie im Beispiel vom Beweis von Satz 3.3 gesehen, fälschlicherweise auch Teilbäume aussortiert, die zwar in der aktuellen Iteration dominiert werden, aber später trotzdem noch effizient sein können.
Ausgehend von dieser Beobachtung korrigieren wir den Algorithmus. Wir berechnen weiterhin in jeder Iteration die entsprechenden Teilbäume und wollen doppelte Teilgraphen vermeiden, da sie einen unnötigen Speicher- und Rechenaufwand bedeuten würden. Deshalb führen wir beim Algorithmus in jeder Iteration explizit das Löschen der doppelten Teilbäume auf.
Erst nach der letzten Iteration sortieren wir alle dominierten Bäume aus. Wir beweisen, dass dieser korrigierte Algorithmus das multikriterielle Minimum Spanning Tree Problem löst, das heißt, dass er alle effizienten Bäume berechnet.

Algorithm 3: Prim's Spanning Tree Algorithmus - Multikriteriell

Input : Einfacher, zusammenhängender, ungerichteter Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Kantengewichte $c\colon \mathcal{E} \to \mathbb{Z}^p$

Output : $\mathcal{T}_{n-1}$, die Menge aller MST von $\mathcal{G}$

1 Bestimme $\mathcal{T}_1 \coloneqq argmin\{c(e_1), \ldots, c(e_m)\}$
2 **for** $k = 2, \ldots, n-1$ **do**
3 Berechne $\mathcal{T}_k \coloneqq \{\, \mathcal{E}(T) \cup \{f\} \,|\, T \in \mathcal{T}_{k-1}\,,\ f \in argmin\{c(e) \,|\, e \in \delta(T)\} \,\}$
4 Lösche doppelte Teilbäume in $\mathcal{T}_k$
5 **end**
6 Setze $\mathcal{T}_{n-1} \coloneqq argmin\{c(T) \,|\, T \in \mathcal{T}_{n-1}\}$
7 **return** $\mathcal{T}_{n-1}$ die Menge aller effizienten spannenden Bäume von $\mathcal{G}$

Um zu beweisen, dass der angepasste Algorithmus korrekt ist, brauchen wir den folgenden Satz aus Ehrgott [8, Cor. 9.20].

Satz 3.4. *Sei T ein effizienter spannender Baum vom Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Dann enthält T eine Kante $e \in argmin\{c(f) \,|\, f \in \mathcal{E}\}$. Eine solche Kante heißt* minimale Kante.

Der folgende Satz zeigt, dass der Algorithmus 3 korrekt ist, also insbesondere alle effizienten spannenden Bäume findet.

Satz 3.5. *Der angepasste Algorithmus 3 ist korrekt.*

Beweis. Wie wir bei Algorithmus 2 gesehen haben, sind die berechneten Teilgraphen effiziente spannende Bäume. Denn sie sind zusammenhängend, haben $n-1$ Kanten und erfüllen die Bedingung 1) aus Satz 3.1. Weil diese Bedingung nur notwendig für die Effizienz ist, werden am Ende alle dominierten Bäume gelöscht. Zu zeigen ist noch, dass alle effizienten spannenden Bäume gefunden werden.
Sei dazu T ein beliebiger effizienter spannender Baum von $\mathcal{G}$ und wir zeigen, dass T in der Rückgabemenge $\mathcal{T}_{n-1}$ enthalten ist.
Nach Satz 3.4 hat T eine minimale Kante. Sei diese $e_1 = [u, v]$. Mit Algorithmus 3 bestimmen wir in Schritt 1 alle minimalen Kanten. Damit gilt

$$e_1 \in \mathcal{T}_1.$$

Nach Lemma 2.7 ist der Schnitt $\delta_T(\{u, v\}) = \delta_T(e_1)$ im Baum T nicht leer und enthält damit eine Kante e_2, welche nach Satz 3.1 1) minimal im Schnitt $\delta_{\mathcal{G}}(T - e_1)$ vom Graphen $\mathcal{G}$ ist.
Für $k = 2$ erzeugen wir beim Algorithmus in Schritt 3 für jede Kante aus $\mathcal{T}_1$ einen neuen Teilbaum durch Hinzufügen einer minimalen Kante aus dem Schnitt. Damit erzeugen wir auch einen Teilbaum durch Hinzufügen von e_2 zu e_1. Somit gilt

$$\{e_1, e_2\} \in \mathcal{T}_2.$$

Analog folgt, dass die verbleibenden $n - 3$ Kanten in der for-Schleife hinzugefügt werden. Denn in jeder Iteration existiert im Schnitt jeweils eine minimale Kante, die wir beim Algorithmus in Schritt 3 bestimmen und zum Teilbaum hinzufügen. Somit gilt nach der letzten Iteration

$$T \in \mathcal{T}_{n-1}.$$

Weil T ein effizienter spannender Baum ist, wird er nicht dominiert und deshalb in Schritt 6 nicht gelöscht. Somit ist T in der Rückgabemenge $\mathcal{T}_{n-1}$ enthalten.
Weil T nach Voraussetzung ein beliebiger effizienter spannender Baum ist, findet der Algorithmus wirklich alle effizienten spannenden Bäume. Somit löst er das multikriterielle Minimum Spanning Tree Problem und ist damit korrekt. □

4 Pareto-Graph und Zusammenhang

Mit Algorithmus 3 haben wir einen korrekten Algorithmus, der das multikriterielle Minimum Spanning Tree Problem löst und somit alle effizienten spannenden Bäume findet.
Dieser Algorithmus ist allerdings für praktische Anwendungen nicht geeignet, da er eine sehr hohe Laufzeit hat. Vor allem das Berechnen der Mengen $\mathcal{T}_k$ in jeder Iteration ist extrem aufwändig, denn diese wächst exponentiell schnell in der Größe der Knoten und Kanten. Sogar für kleine Graphen mit $|\mathcal{V}(\mathcal{G})| = 25$ Knoten werden zum Teil sehr große Mengen von Teilgraphen berechnet. Siehe dazu auch Kapitel 5.3 bezüglich eigener Beobachtungen und Auswertungen des in C++ implementierten Programms.
Dies motiviert die Suche nach Alternativen zur Berechnung der effizienten Menge beim multikriteriellen Minimum Spanning Tree Problem. Eine Möglichkeit bietet eine Nachbarschaftssuche (engl.: Neighborhood Search). Hierbei erzeugt man durch ein einfaches Verfahren einen minimal spannenden Baum T, zum Beispiel durch lexikographische Suche.
Durch den Austausch von Kanten sucht man nach neuen effizienten spannenden Bäumen in der "Nachbarschaft" von T. Dabei würde man nur einen solchen Austausch von zwei Kanten $e \in \mathcal{E}(T)$ und $f \in \mathcal{E}(\mathcal{G})\backslash\mathcal{E}(T)$ betrachten, für den $c(e) - c(f)$ positive und negative Komponenten hat, da verschiedene effiziente spannende Bäume nichtvergleichbare Kostenvektoren haben müssen.
Für diese Nachbarschaftssuche gibt es bereits viele effizient arbeitende Algorithmen, siehe zum Beispiel Andersen u. a. [1], Hamacher und Ruhe [14] oder Shioura u.a. [20]. Die Frage ist allerdings, ob man mit diesem Verfahren alle effizienten Lösungen bestimmen kann.
Dafür beschreiben wir den Zusammenhang von $\mathcal{T}_E$ graphentheoretisch mit Hilfe des sogenannten Pareto-Graphen und gehen später auf die spezielle Situation im bikriteriellen Fall ein. Die folgenden Definitionen sind angelehnt an Ehrgott [8] und Gorski u. a. [12].

4.1 Zusammenhang der effizienten Menge

Wir definieren zuerst, wann zwei spannende Bäume adjazent sind, und darauf aufbauend den Zusammenhang der effizienten Menge $\mathcal{T}_E$.

Definition 4.1 (Adjazenz von spannenden Bäumen)**.**
Zwei spannende Bäume T_1 und T_2 heißen *adjazent* oder *benachbart*, falls sie $n-2$ Kanten gemeinsam haben.

Bemerkung. Zwei spannende Bäume T_1, T_2 sind daher adjazent, wenn sie sich in genau einer Kante unterscheiden. Wir sagen auch, dass sie *zusammenhängend* sind.

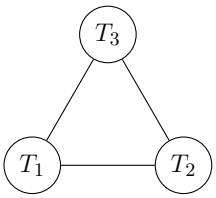
Wir führen den Pareto-Graph ein, wie bei Ehrgott [7] definiert.

Definition 4.2 (Pareto-Graph)**.**
Sei $\mathcal{G}$ ein ungerichteter Graph. Dann ist der zugehörige *Pareto-Graph* definiert als ein ungerichteter Graph $\mathcal{P}(\mathcal{G}) = (\mathcal{V}(\mathcal{P}(\mathcal{G})), \mathcal{E}(\mathcal{P}(\mathcal{G})))$ mit

- $\mathcal{V}(\mathcal{P}(\mathcal{G})) = \mathcal{T}_E$
- $[T_1, T_2] \in \mathcal{E}(\mathcal{P}(\mathcal{G}))$, falls T_1 und T_2 adjazent sind im Sinne von Definition 4.1.

Der Pareto-Graph hat somit für jeden effizienten spannenden Baum einen Knoten und es existiert genau dann eine Kante zwischen zwei Knoten, wenn die zugehörigen Bäume adjazent sind.

Beispiel 4.3. Der Graph des Gegenbeispiels im Beweis von Satz 3.3 hat drei effiziente spannende Bäume T_1, T_2 und T_3, die sich jeweils nur in einer Kante unterscheiden. Damit erhalten wir folgenden Pareto-Graphen:



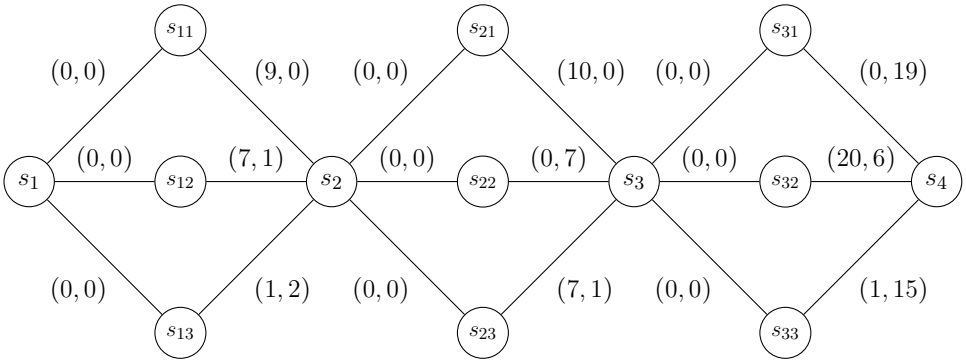
Ausgehend von Definition 4.2 definieren wir den Zusammenhang der effizienten Menge über den Zusammenhang des Pareto-Graphen, siehe auch Gorski u. a. [12].

Definition 4.4. Die effiziente Menge $\mathcal{T}_E$ ist genau dann *zusammenhängend*, wenn der zugehörige Pareto-Graph $\mathcal{P}(\mathcal{G})$ zusammenhängend ist.

Die Neighborhood Search findet demnach genau dann alle effizienten spannenden Bäume, wenn die zugehörige effiziente Menge $\mathcal{T}_E$ beziehungsweise der zugehörige Pareto-Graph $\mathcal{P}(\mathcal{G})$ zusammenhängend ist.

Für das multikriterielle Minimum Spanning Tree Problem gibt es allerdings schon bekannte Beispiele, dass dies im Allgemeinen nicht der Fall ist. Betrachte folgendes Beispiel von Ehrgott und Klamroth [9].

Beispiel 4.5. Gegeben sei folgender Graph $\mathcal{G}$



Jeder effiziente spannende Baum enthält alle Kanten mit Gewicht $(0,0)$ und zusätzlich jeweils genau eine der drei Kanten $[s_{i,j}, s_{i+1}]$, $j = 1,2,3$, welche die Knoten s_{i+1}, $i = 1,2,3$ mit den links liegenden Knoten verbindet. Es existieren zwölf effiziente spannende Bäume, welche in Tabelle 1 aufgezählt sind, wobei die Kanten mit Kosten $(0,0)$ weggelassen wurden.

EST	Kanten	Gewicht
T_1	$[s_{13}, s_2][s_{22}, s_3][s_{31}, s_4]$	$(1,28)$
T_2	$[s_{13}, s_2][s_{22}, s_3][s_{33}, s_4]$	$(2,24)$
T_3	$[s_{13}, s_2][s_{23}, s_3][s_{31}, s_4]$	$(8,22)$
T_4	$[s_{13}, s_2][s_{23}, s_3][s_{33}, s_4]$	$(9,18)$
T_5	$[s_{13}, s_2][s_{21}, s_3][s_{33}, s_4]$	$(12,17)$
T_6	$[s_{11}, s_2][s_{23}, s_3][s_{33}, s_4]$	$(17,16)$
T_7	$[s_{11}, s_2][s_{21}, s_3][s_{33}, s_4]$	$(20,15)$
T_8	$[s_{12}, s_2][s_{22}, s_3][s_{32}, s_4]$	$(27,14)$
T_9	$[s_{13}, s_2][s_{21}, s_3][s_{32}, s_4]$	$(28,9)$
T_{10}	$[s_{13}, s_2][s_{21}, s_3][s_{32}, s_4]$	$(31,8)$
T_{11}	$[s_{11}, s_2][s_{23}, s_3][s_{32}, s_4]$	$(36,7)$
T_{12}	$[s_{11}, s_2][s_{21}, s_3][s_{32}, s_4]$	$(39,6)$

Tabelle 1: Effiziente spannende Bäume des Graphen $\mathcal{G}$

In der folgenden Abbildung 1 sehen wir den zugehörigen Pareto-Graphen $\mathcal{P}(\mathcal{G})$. Dieser wurde mit dem in C++ implementierten Programm berechnet und mit

Hilfe des Graph File Editors *GVEdit v1.01* von *Graphviz v2.28* gezeichnet. Siehe dazu auch Kapitel 5.

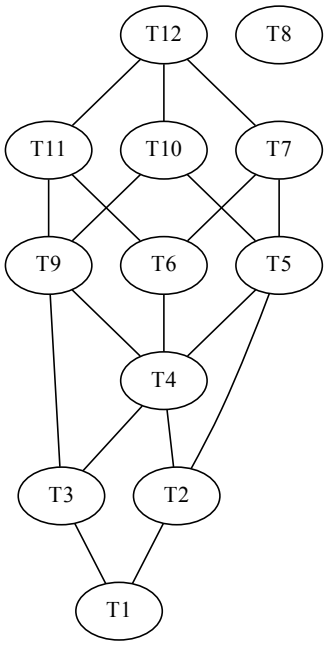


Abbildung 1: Zugehöriger Pareto-Graph $\mathcal{P}(\mathcal{G})$

Wir erkennen, dass der Knoten $T8$ isoliert ist. Tabelle 1 bestätigt, dass der zugehörige effiziente spannende Baum T_8 sich in mindestens zwei Kanten von allen anderen Bäumen unterscheidet. Demnach ist der Pareto-Graph und somit die effiziente Menge in diesem Beispiel nicht zusammenhängend. Eine Nachbarschaftssuche würde in diesem Fall nicht alle effizienten spannenden Bäume finden.

Bemerkung. Es gibt weitere bekannte Gegenbeispiele, siehe zum Beispiel Andersen u. a. [1].

Ehrgott und Klamroth [9] bewiesen darüber hinaus, dass jeder Graph erweitert werden kann, so dass dessen Pareto-Graph nicht zusammenhängend ist. Dies besagt der folgende Satz, siehe auch Ehrgott und Klamroth [9].

Satz 4.6. *Für jeden Graph $\mathcal{G}$ existiert ein Graph $\mathcal{G}^*$, der $\mathcal{G}$ als Teilgraph enthält und für den gilt, dass sein Pareto-Graph $\mathcal{P}(\mathcal{G}^*)$ nicht zusammenhängend ist.*

Trotz Satz 4.6 gibt es Klassen von Graphen, bei denen der zugehörige Pareto-Graph zusammenhängend ist. Für ein Beispiel betrachte folgendes Lemma.

Lemma 4.7. *Sei $\mathcal{G}$ ein* 1-tree*, auch* Pseudo-Baum *genannt, das heißt ein Baum mit genau einer zusätzlichen Kante beziehungsweise ein zusammenhängender Graph mit genau einem Kreis. Dann ist der zugehörige Pareto-Graph $\mathcal{P}(\mathcal{G})$ immer zusammenhängend.*

Beweis. Man erhält alle spannenden Bäume eines $1-$tree, indem man jeweils eine Kante aus dem Kreis herausnimmt. Demnach unterscheiden sich zwei verschiedene effiziente spannende Bäume eines $1-$tree in genau einer Kante und sind somit adjazent. Damit ist der zugehörige Pareto-Graph zusammenhängend. □

Wie wir gesehen haben, ist eine Nachbarschaftssuche nur bei einer zusammenhängenden effizienten Menge möglich, was im Allgemeinen nicht gegeben ist. Demnach sind Neighborhood Search Algorithmen nur bei bestimmten Klassen von Graphen mit zusammenhängenden Pareto-Graphen, zum Beispiel Pseudo-Bäumen, eine sinnvolle Alternative zum Algorithmus 3.
Im Folgenden untersuchen wir, welche Auswirkungen Restriktionen der Kostenfunktion auf den Pareto-Graphen haben. Dabei beschränken wir uns auf das bikriterielle Minimum Spanning Tree Problem.

4.2 Bikriterieller Fall

Beim bikriteriellen Minimum Spanning Tree Problem (BMSTP) betrachten wir eine ganzzahlige Kostenfunktion mit zwei Komponenten, also

$$c\colon \mathcal{E} \to \mathbb{Z}^2.$$

Beispielsweise kann man sich eine Spedition vorstellen, die verschiedene Standorte mit LKWs hat, eine minimale Verbindung zwischen ihnen sucht und die Lösungen nur nach den Kriterien Entfernung und Geschwindigkeit bewertet.
Es wurde schon 1994 von Hamacher und Ruhe [14] entdeckt, dass das bikriterielle MSTP *intractable* ist, das heißt, dass die Menge der optimalen Lösungen im worst-case exponentiell in der Größe der gegebenen Instanz ist. Betrachte dazu folgenden Satz.

Satz 4.8. *Das bikriterielle Minimum Spanning Tree Problem ist intractable.*

Der Beweis des Satzes benutzt die Formel von Cayley [3], welche besagt, dass ein vollständiger Graph mit n Knoten genau n^{n-2} spannende Bäume besitzt. Eine Folgerung des Satze ist, dass es keinen Algorithmus geben kann, der in polynomieller Zeit alle effizienten Lösungen berechnet.
Zusätzlich haben wir es mit einem $\mathcal{NP}$-vollständigem Problem zu tun, wie Camerini u. a. [2] zeigten:

Satz 4.9. *Das bikriterielle Minimum Spanning Tree Problem ist $\mathcal{NP}$-vollständig.*

Trotz der negativen Resultate aus Satz 4.8 und Satz 4.9 gibt es auf dem Gebiet des bikriteriellen Minimum Spanning Tree Problems bereits einige vielversprechende Erkenntnisse und Fortschritte. Schränkt man die zweite Komponente des Kostenvektors auf 0 oder 1 ein, also auf binäre Kosten, dann ist der Pareto-Graph immer zusammenhängend. Vorstellbar sind solche Kosten zum Beispiel als Auswahlkriterium, wobei die 0 für die favorisierte Entscheidung steht. Dies zeigt der folgende Satz, zu finden in der Dissertation von Gorski [11].

Satz 4.10. *Sei eine zulässige Instanz des bikriteriellen Minimum Spanning Tree Problem mit binären Kosten in der zweiten Komponente gegeben. Dann ist die zugehörige effiziente Menge $\mathcal{T}_E$ zusammenhängend.*

Beweis. Der Satz wurde von Gorski [11, Kor. 10.15] für das *biobjective matroid problem with binary costs* (BBMP) bewiesen. Insbesondere ist $(\mathcal{E},\mathcal{U})$ ein Matroid, wobei $\mathcal{E} = \mathcal{E}(\mathcal{G})$ die Kantenmenge des gegebenen Graphen ist und $\mathcal{U}$ die Menge aller kreisfreien Teilmengen dieser Kanten. Da wir nur zusammenhängende Graphen betrachten, sind nach Satz 2.8 f) die spannenden Bäume die maximalen Elemente in $\mathcal{U}$. Somit ist $\mathcal{T}$, die Menge der spannenden Bäume, in $\mathcal{U}$ enthalten. Damit folgt die Behauptung. □

Der Satz 4.10 motiviert weitere Untersuchungen auf dem Gebiet des bikriteriellen Minimum Spanning Tree Problems. Im nächsten Kapitel betrachten wir den Spezialfall, bei welchem wir die zweite Kostenkomponente von binären Kosten um den zusätzlichen Wert 2 auf $\{0, 1, 2\}$ erweitern. Wir untersuchen, ob die effiziente Menge hierbei zusammenhängend bleibt.

5 Experimente

5.1 Motivation

Durch Einschränken der Graphen auf die Klasse der Pseudo-Bäume oder auf das bikriterielle Minimum Spanning Tree Problem mit binären Kosten in der zweiten Komponente sind nach Lemma 4.7 und Satz 4.10 die erhaltenen effizienten Mengen $\mathcal{T}_E$ immer zusammenhängend. Dies legitimiert erstmals die Anwendung eines Neighborhood Search Algorithmus, um alle effizienten Lösungen dieser restringierten Probleme zu bestimmen.
Es stellt sich die Frage, ob es weitere Klassen von Graphen oder Einschränkungen der Kosten gibt, so dass die effiziente Menge immer noch zusammenhängend bleibt. Ausgehend von Satz 4.10 erweitern wir in dieser Arbeit das bikriterielle Minimum Spanning Tree Problem mit binären Kosten in der zweiten Kostenkomponente um den zusätzlichen Wert 2.
Es handelt sich um ein offenes Problem, ob bei diesem restringierten bikriteriellen Minimum Spanning Tree Problem mit Werten aus $\{0, 1, 2\}$ in der zweiten Kostenkomponente die effiziente Menge zusammenhängend ist und somit eine Nachbarschaftssuche alle effizienten spannenden Bäume finden kann.
Vor allem bikriterielle Kosten bieten sich für Experimente und Untersuchungen an, da diese sehr übersichtlich und relativ leicht zu berechnen sind. Zusätzlich hat dieses restringierte Problem mit eingeschränkten Kosten in der zweiten Komponente einen hohen Praxisbezug.
Anwendungsbeispiele sind für diverse Problemstellungen vorstellbar, bei denen Kosten minimiert und zugleich drei verschiedene zusätzliche Auswahlkriterien beachtet werden sollen, wobei Kosten 0 in der zweiten Komponente die “favorisierte” und 2 die “zu vermeidende” Entscheidung repräsentieren.
Als Beispiel betrachten wir einen Energiekonzern, welcher das Stromnetz plant. Neue Hochspannungsleitungen müssen die Grundversorgung sichern und gleichzeitig sollen die Kosten für den Bau möglichst gering sein. Allerdings möchte man Freileitungstrassen in bestimmten Räumen wie Naturschutz-, Siedlungs- oder Erholungsgebieten vermeiden. Denkbar ist dabei eine dreistufige Bewertung von Gebieten. Durch Bestimmen der effizienten spannenden Bäume können verschiedene Lösungen verglichen werden. Werte nahe bei 0 in der zweiten Komponente bedeuten, dass man unerwünschte Gebiete umgeht, zugleich können aber höhere

Kosten durch in Kauf genommene Umwege entstehen. Andererseits sind durch den Bau in Siedlungsnähe oder in anderen Gebieten mit “negativer Wertung” Kosteneinsparungen denkbar.
Im Folgenden wollen wir das offene Problem des Zusammenhangs numerisch lösen. Hierfür entwickeln wir ein Programm in C++, welches zufällige Testgraphen erzeugt und die zugehörigen Pareto-Graphen auf Zusammenhang testet.

5.2 Implementation

Für das Erzeugen und Speichern von Graphen brauchen wir eine geeignete Datenstruktur. Hierfür verwenden wir sogenannte Adjazenzmatrizen:

Definition 5.1 (Adjazenzmatrix)**.**
Die *Adjazenzmatrix* $AD = AD(\mathcal{G})$ eines ungerichteten Graphen $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ mit $|\mathcal{V}| = n$ ist die $n \times n$-Matrix mit

$$a_{ij} = |\{e \in \mathcal{E} \,|\, \gamma(e) = \{i, j\}\}|.$$

Bemerkung. Bei der Adjazenzmatrix steht der Eintrag a_{ij} für die Anzahl der Kanten, welche die Knoten i und j verbinden. Für einfache Graphen $\mathcal{G}$ erhalten wir als Adjazenzmatrix $AD(\mathcal{G})$ demnach eine 0-1-Matrix, das heißt, dass die Einträge von $AD(\mathcal{G})$ nur aus den Werten 0 oder 1 bestehen.

Da wir beim multikriteriellen Minimum Spanning Tree Problem nur einfache Graphen betrachten, können wir die Einträge der Adjazenzmatrix $AD(\mathcal{G})$ zum Speichern der Kosten auf den entsprechenden Kanten verwenden. Dabei ersetzen wir den Eintrag 1 für eine Kante $e = [v_i, v_j]$ an der Stelle a_{ij} durch ihr Gewicht $c(e)$. Bei der Implementation müssen wir lediglich auf Kanten mit Kosten 0 achten, beispielsweise durch Addieren von +1 auf alle Kantengewichte, um sie identifizieren zu können.
Der folgende Algorithmus erzeugt in Abhängigkeit von der vorgegebenen Knotenanzahl n und eines Maximalgewichts MaxGew einen zufälligen, gewichteten und zusammenhängenden Graphen. Diesen werden wir unter der Benutzung von Adjazenzmatrizen für die Implementation in C++ verwenden.

Algorithm 4: Zufälligen, gewichteten, zusammenhängenden Graph erzeugen

Input : Knotenanzahl $n \in \mathbb{N}$, Maximalgewicht MaxGew

Output : Zufälliger, gewichteter, zusammenhängender Graph $\mathcal{G}$

1 Erzeuge n Knoten $v_0, \ldots, v_{n-1}$
2 Setze $\mathcal{V}(\mathcal{G}) \coloneqq \{v_0, \ldots, v_{n-1}\}$ und $\mathcal{E}(\mathcal{G}) \coloneqq \emptyset$
3 **for** $i = 1, \ldots, n-1$ **do**
4 $j = \text{random}()\%i$
5 Erzeuge Kante $e_i = [v_j, v_i]$ mit Kosten $c(e_i) = \text{random}()\%\text{MaxGew}$
6 $\mathcal{E}(\mathcal{G}) = \mathcal{E}(\mathcal{G}) \cup \{e_i\}$
7 **end**
8 **return** $\mathcal{G}$

Bemerkung. Wie in Programmiersprachen üblich erzeugt die Funktion random() eine zufällige natürliche Zahl und der %-Operator steht für Modulo.

Der Algorithmus erzeugt am Anfang n Knoten. In der for-Schleife verbindet er die ersten beiden Knoten v_0 und v_1, und danach jeden weiteren Knoten v_i mit einem zufälligen Knoten v_j mit $j < i$. Auf den Kanten werden außerdem zufällige Gewichte zwischen $0, \ldots, \text{MaxGew}-1$ generiert.

Satz 5.2. *Der Algorithmus 4 ist korrekt.*

Beweis. Zu zeigen ist, dass der erzeugte Graph $\mathcal{G}$ wirklich zusammenhängend ist. Wir beweisen dies per vollständiger Induktion über die Knotenanzahl n:

IA: $n = 2$

Für $n = 2$ werden zwei Knoten v_0 und v_1 erzeugt, und die for-Schleife genau einmal ausgeführt. In Schritt 4 gilt $j =$random()$\%1 = 0$. Demnach wird in Schritt 6 eine Kante $e_1 = (v_0, v_1)$ erstellt, wodurch beide Knoten verbunden werden. Somit sind sie zusammenhängend.

IV: Gelte die Behauptung für alle $k \leq n$ mit $k \in \mathbb{N}$ und n fest.

IS: $n \mapsto n + 1$

Der zusätzlich erzeugte Knoten v_n wird in der Iteration $i = n$ mit einem zufälligen Knoten v_j, $j \in \{0, \ldots, v_{n-1}\}$ verbunden. Die Knoten $v_0, \ldots, v_{n-1}$ sind nach IV schon zusammenhängend. Damit folgt die Behauptung. □

Wir implementieren Algorithmus 4, um einen zufälligen zusammenhängenden Graphen mit bikriteriellen Gewichten zu erzeugen. Dafür erstellen wir in C++ eine $n \times n \times 2$ Adjazenzmatrix AD, deren Einträge wir in Schritt 5 des Algorithmus wie folgt bestimmen:

$$AD[j][i][0] = \text{random}()\%\text{MaxGew} + 1 \quad \text{und} \quad AD[j][i][1] = \text{random}()\%3.$$

Beide Komponenten zusammen bilden unsere bikriteriellen Gewichte, wobei die erste Komponente zum Identifizieren der Kante +1 gerechnet wird.
Mit Algorithmus 4 erhalten wir allerdings nur einen minimal zusammenhängenden Graphen, weshalb wir in einer weiteren for-Schleife, wie oben, zusätzlich $m \in \mathbb{N}$ zufällige Einträge in der Matrix AD erzeugen. Diese können sich auch gegenseitig überschreiben. Beim so erhaltenen Graphen können wir damit wirklich von einem zufälligen Graphen sprechen.
Die effizienten spannenden Bäume dieses Testgraphen bestimmen wir durch Implementieren des Algorithmus 3, der korrigierten multikriteriellen Version von Prim's Algorithmus. Bei diesem Algorithmus berechnen wir in jeder Iteration in Schritt 3 eine Menge von Teilbäumen $\mathcal{T}_k$. Deshalb erstellen wir Klassen für Kanten, Knoten und Graphen, mit deren Hilfe die Speicherung und das Zugreifen auf Methoden leichter ist als bei Adjazenzmatrizen. Diese orientieren sich an bereits implementierten Klassen aus der Vorlesung *Netzwerkflüsse* von Westphal [22].
Im Anschluss bestimmen wir aus der erhaltenen effizienten Menge den zugehörigen Pareto-Graphen. Dazu legen wir einen neuen Graphen $\mathcal{P}$ an, indem wir für jeden effizienten spannenden Baum einen Knoten erzeugen und genau dann eine Kante zwischen zwei Knoten erstellen, wenn die zugehörigen Bäume adjazent sind.
Als letztes überprüfen wir, ob der Pareto-Graph und damit die effiziente Menge zusammenhängend ist. Dafür verwenden wir einen Graph Scanning Algorithmus. Der Algorithmus mit Beweis kann zum Beispiel bei Korte und Vygen [16] nachgelesen werden. Wir passen ihn an, indem wir auf ein zusätzliches Speichern und die Rückgabe von Kanten verzichten, denn uns interessieren nur die Knoten, die von einem Startknoten s aus erreichbar sind und damit in einer Zusammenhangskomponente liegen.

Algorithm 5: Graph Scanning Algorithm - Angepasste Version

Input : $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ zusammenhängend, einfach, $s \in \mathcal{V}$
Output : Knotenmenge $R \subseteq \mathcal{V}$, der von s aus erreichbaren Knoten

```
1  R = {s}, Q = {s}
2  while Q ≠ ∅ do
3      Wähle beliebiges v ∈ Q
4      if ∃ w ∈ V\R mit [v, w] ∈ E then
5          R = R ∪ {w}, Q = Q ∪ {w}
6      else
7          Q = Q\{v}
8      end
9  end
10 return R
```

Der Algorithmus erzeugt zwei Mengen R und Q, die anfangs beide den Startknoten s enthalten. Solange die Menge Q nichtleer ist, wird ein beliebiger Knoten v aus Q gewählt und überprüft, ob ein adjazenter Knoten w existiert, der noch nicht der Menge R hinzugefügt wurde. Falls ja, wird der Knoten w zu beiden Mengen R und Q hinzugefügt, ansonsten wird v aus der Menge Q gelöscht.
Der Algorithmus gibt die Menge R zurück, die alle Knoten enthält, die vom Startknoten s aus erreichbar sind. Ein Graph $\mathcal{G}$ ist demnach genau dann zusammenhängend, wenn der Algorithmus $R = \mathcal{V}(\mathcal{G})$ zurückgibt.
Wir implementieren den Algorithmus in C++, indem wir zusätzlich eine Map anlegen, in welcher wir für jeden Knoten speichern, ob er der Menge R hinzugefügt wurde. Anschließend überprüfen wir in einer Schleife, ob alle Knoten hinzugefügt wurden und geben das Ergebnis aus.

5.3 Beobachtungen

Das Programm wurde so implementiert, dass der Benutzer die Knotenanzahl n, eine "Richtlinie" für die Kantenanzahl m und einen positiven Wert für das maximal mögliche Kantengewicht angibt. Beim Erzeugen der zufälligen Graphen und deren zugehörigen Pareto-Graphen wurden diese in den Dateien *graph.dot*

und *pareto.dot* in der dot-Sprache gespeichert und können somit vom Graph File Editor *GVEdit v1.01* von *Graphviz v2.28* eingelesen und als visueller Graph ausgegeben werden.
In die oben beschriebene Implementation wurde eine while-Schleife eingebunden, die solange Graphen erzeugt und untersucht, bis ein nicht zusammenhängender Pareto-Graph gefunden wird.
Doch selbst nach zigfacher Ausführung des Programms, selbst nach weit mehr als zehn Millionen untersuchten zufälligen Graphen, konnte bisher kein Gegenbeispiel gefunden werden. Dabei wurden verschiedene Knotenanzahlen $n = 3, \ldots, 50$ getestet und das Maximalgewicht zwischen 3 bis 100.000 variiert. Zusätzlich wurden folgende Werte für m gewählt:

$$\left\lfloor \frac{n}{4} \right\rfloor , \left\lfloor \frac{n}{2} \right\rfloor , \left\lfloor \frac{n \cdot n}{4} \right\rfloor \text{ oder } n^2.$$

Man beachte, dass ein vollständiger Graph mit n Knoten nur $\frac{n(n-1)}{2}$ Kanten hat. Allerdings werden beim Erzeugen der zufälligen Kanten Einträge in der Adjazenzmatrix generiert, die sich gegenseitig überschreiben können. Daher wurde auch eine Funktion zum Erzeugen von vollständigen Graphen mit zufälligen Gewichten implementiert und getestet.
Da mit Hilfe dieser Einstellungen kein Gegenbeispiel gefunden werden konnte, wurde sich an Beispiel 4.5 orientiert. Der Graph hat sehr viele Kanten mit Gewicht $(0, 0)$, welche sich in jedem minimal spannenden Baum wiederfinden. Deshalb wurde bei der Implementation eine zusätzliche Variable *nullen* als Richtlinie für zufällige Kanten mit Gewicht $(0, 0)$ eingebunden, wobei folgende Werte getestet wurden:

$$\left\lfloor \frac{n}{8} \right\rfloor , \left\lfloor \frac{n}{6} \right\rfloor , \left\lfloor \frac{n}{4} \right\rfloor \text{ oder } \left\lfloor \frac{n}{2} \right\rfloor .$$

Außerdem wurde das Maximalgewicht auf 100 gesetzt. Doch auch hier wurden nach mehrfacher Ausführung nur zusammenhängende Pareto-Graphen beobachtet.
Im Anhang befinden sich Beispiele der von *GVEdit* gezeichneten Pareto-Graphen. Für kleine n, also Graphen zwischen 3 bis 10 Knoten, erhalten wir meist auch kleine Pareto-Graphen mit 1 bis 5 Knoten. Für große n zwischen 10 bis 50 variiert die Größe der Pareto-Graphen sehr stark. Auch hier wurden Pareto-Graphen mit nur einem Knoten beobachtet, allerdings auch Fälle mit weit mehr als 100 Knoten. Wie schon erwähnt, sind dabei alle berechneten Pareto-Graphen zusammenhängend.

Deren Knoten sind in einigen Fällen nur zu einer Kante inzident, bei anderen wiederum existieren für alle Knoten jeweils mehrere ausgehende Kanten. Eine bezeichnende Struktur der Pareto-Graphen kann demnach nicht identifiziert werden.
Die Laufzeit des Programms pro zufällig erzeugtem Graph schwankt sehr stark. In einigen Fällen beträgt diese für große Graphen mit $n = 50$ Knoten nur einige Sekunden, in anderen Fällen wiederum bricht das Programm ab, weil zu viel Speicher für das Berechnen der Mengen $\mathcal{T}_k$ benötigt wurde. Die Anzahl der berechneten Teilbäume ist dabei ein entscheidender Faktor. Wir erwarten nach Satz 4.8, dass diese Anzahl exponentiell von der Anzahl der Knoten abhängt.
Dazu wurden im Folgenden für $n = 10, \ldots, 35$ und für ein Maximalgewicht von 100 jeweils 100 zufällige Graphen mit wenigen Kanten, das heißt $m = \frac{n}{2}$ und $nullen = \frac{n}{6}$, erzeugt. Für festes n wurde für jeden der 100 Graphen die Anzahl der berechneten Teilbäume, das heißt die Summe $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ berechnet und von den erhaltenen Werten das Minimum, das Maximum und der Mittelwert bestimmt.
Die beobachteten Daten befinden sich im Anhang (Tabelle 2). Die Ergebnisse sind in Abhängigkeit von der Knotenanzahl n in der Abbildung 2 graphisch dargestellt.

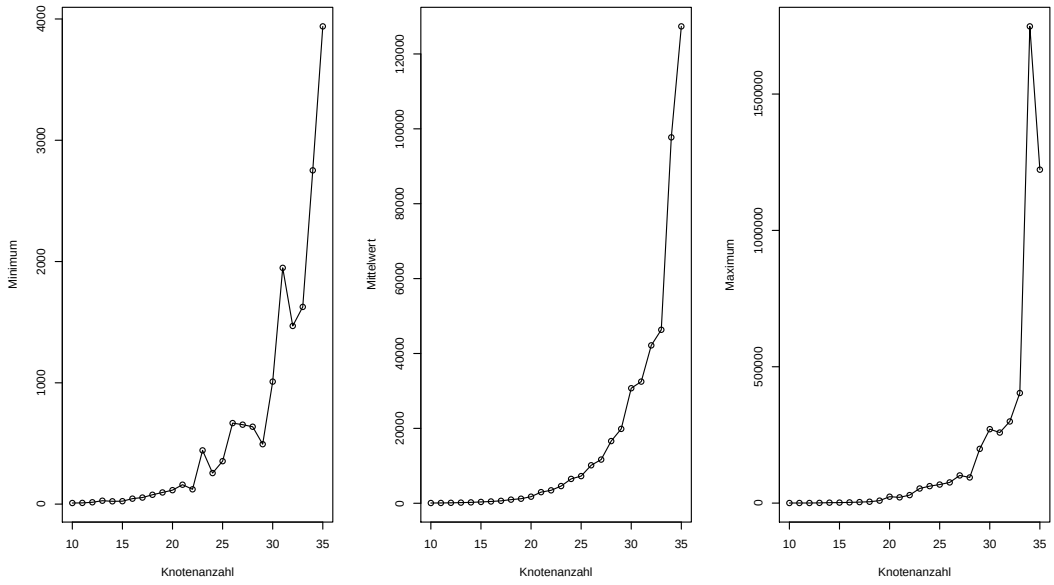


Abbildung 2: $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ in Abhängigkeit von n bei jeweils 100 zufällig erzeugten Graphen mit Maximalgewicht 100 und wenigen Kanten ($m = n/2$, $nullen = n/6$)

Wir erkennen, dass die Anzahl der berechneten Teilgraphen exponentiell von der Anzahl der Knoten n abzuhängen scheint. Allerdings sind zum Teil sehr starke Schwankungen vorhanden, vor allem beim Minimum und beim Maximum. Diese können jedoch auch auf die kleine Testgröße von 100 Durchläufen zurückzuführen sein.
Beachtlich ist, dass schon bei der Knotenanzahl $|\mathcal{V}(\mathcal{G})| = 25$ im Mittel mehr als 7000 und als Maximum sogar mehr als 68.000 Teilgraphen berechnet und gespeichert wurden. Um $n = 35$ werden sogar Spitzenwerte von über einer Million Teilbäume erreicht.
Zusätzlich wurde diese Auswertung für mittel viele Kanten durchgeführt (Tabelle 3). Analog wie oben wurden jeweils 100 zufällige Graphen mit $m = \frac{n \cdot n}{2}$ und $nullen = \frac{n}{4}$ erzeugt, die Anzahl der berechneten Teilgraphen $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ bestimmt und die minimale, maximale und durchschnittliche Anzahl gespeichert. Allerdings wurden hier nur Knotenanzahlen von $n = 10, \ldots, 25$ getestet, denn wie man in der Abbildung 3 erkennt, scheint auch die Knotenanzahl sich stark auf die Anzahl der zu berechnenden Teilbäume auszuwirken.

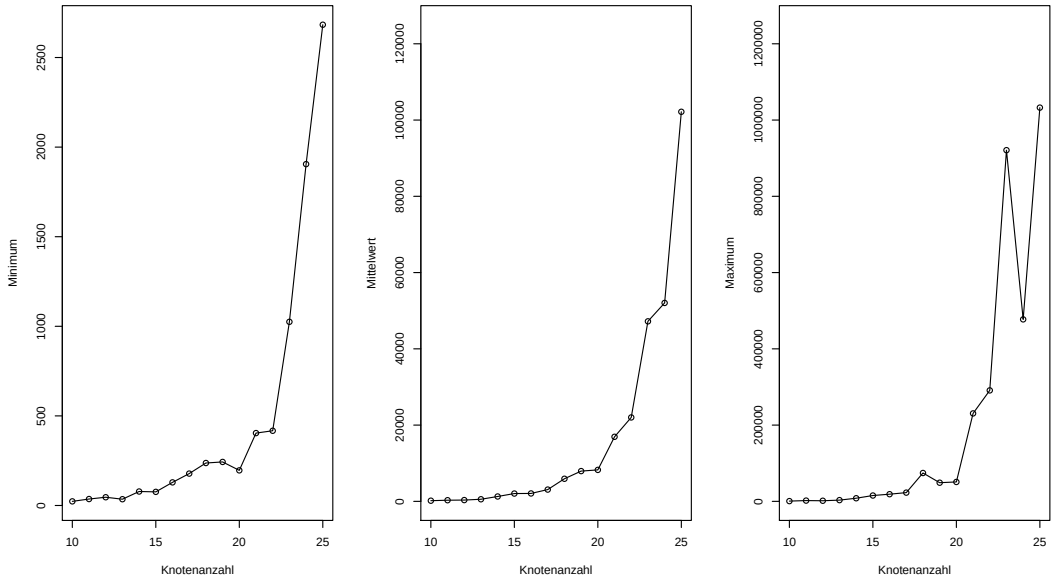


Abbildung 3: $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ in Abhängigkeit von n bei jeweils 100 zufällig erzeugten Graphen mit mittel vielen Kanten ($m = {}^{n \cdot n}/_{2}$, $nullen = {}^{n}/_{4}$)

Auch hier scheinen die Werte exponentiell von n abzuhängen. Wahrscheinlich be-

steht zusätzlich eine Abhängigkeit von der Kantenanzahl m, denn mit diesen leicht erhöhten Kantenanzahlen berechnen wir bereits für $|\mathcal{V}(\mathcal{G})| = 25$ im Durchschnitt über Hunderttausend, als Maximalwert sogar über eine Million Teilgraphen.
Die Sprünge und Abweichungen in den beiden Abbildungen sowie die sehr großen Differenzen zwischen Maxima und Minima erklären die beobachteten unterschiedlichen Laufzeiten. Außerdem hängt die Laufzeit des Algorithmus 3 wahrscheinlich exponentiell von der Anzahl der Knoten und Kanten ab.
Dies bestätigt noch einmal, dass der Algorithmus für die Praxis nicht geeignet ist. Zum einen können die Ergebnisse sogar für kleine Graphen praktisch nicht von Hand nachgerechnet werden, zum anderen ist der Speicher- und Rechenaufwand sowie die damit verbundene Laufzeit oftmals zu hoch.
Vergleiche hierzu auch den Beispielgraph in Abbildung 15 aus dem Anhang. Hier wurden für $n = 50$ Knoten und wenig bis mittel viele Kanten mehr als 2,5 Millionen Teilgraphen berechnet.
Die Abbildungen 2 und 3 dienen allerdings nur dazu, einen ungefähren Überblick über die Anzahl der berechneten Teilgraphen zu erhalten. Die Testgröße mit 100 zufällig erzeugten Graphen ist für statistisch aussagekräftige Ergebnisse zu klein. Außerdem beeinflussen Ausreißer, vor allem die sehr großen Werte der Maxima, die durchschnittlichen Werte sehr stark. Bei den meisten Graphen werden weniger Teilgraphen berechnet. Da wir allerdings in einer for-Schleife das Programm mehrfach ausführen, sind diese Durchschnittswerte wiederum sinnvoll für die Auswertung.

6 Zusammenfassung und Ausblick

Zu Beginn dieser Arbeit haben wir in Kapitel 2 die Grundlagen der Graphentheorie, des Minimum Spanning Tree Problems und der multikriteriellen Optimierung kennengelernt.
Darauf aufbauend haben wir in Kapitel 3 das multikriterielle Minimum Spanning Tree Problem eingeführt und mit Satz 3.1 notwendige Bedingungen für effiziente spannende Bäume bewiesen, worauf die multikriterielle Version von Prim's Algorithmus von Hamacher und Ruhe [14] basiert. Wir haben in einem Gegenbeispiel gezeigt, dass dieser Algorithmus nicht alle effizienten spannenden Bäume findet und eine modifizierte Version angegeben, deren Korrektheit wir in Satz 3.5 bewiesen haben. Der entstandene Algorithmus 3 ist allerdings aufgrund der hohen Laufzeit für die Praxis nicht geeignet.
Eine Alternative zur Berechnung der effizienten Menge wäre eine Nachbarschaftssuche, welche jedoch nur bei einer zusammenhängenden effizienten Menge möglich ist. Deshalb haben wir in Kapitel 4 den Pareto-Graph und den Zusammenhang der effizienten Menge $\mathcal{T}_E$ definiert.
Wie Ehrgott und Klamroth [9] zeigten, ist der Pareto-Graph im Allgemeinen nicht zusammenhängend und jeder Graph kann sogar erweitert werden, so dass die Neighborhood Search nicht alle Lösungen findet.
Aufgrund dessen haben wir die Klasse der Pseudo-Bäume und das bikriterielle Minimum Spanning Tree Problem mit binären Kosten als Beispiele für Graphen mit zusammenhängender effizienter Menge aufgeführt. Motiviert durch diese Erkenntnisse haben wir in Kapitel 5 Algorithmen zum zufälligen Erzeugen von zusammenhängenden Graphen, den multikriteriellen Algorithmus von Prim und einen Graph-Scanning-Algorithmus in C++ programmiert, um die effiziente Menge beim Erweitern der binären Kosten auf Werte aus $\{0, 1, 2, \}$ auf Zusammenhang zu überprüfen.
Wir haben das Programm mit mehreren verschiedenen Einstellungen für die Knoten- und Kantenanzahl versehen und ausführlich getestet. Doch sogar nach mehr als zehn Millionen erzeugten zufälligen Graphen konnte kein nichtzusammenhängender Pareto-Graph beobachtet werden. Zusätzlich wurde der hohe Rechen- und Speicheraufwand des implementierten Algorithmus anhand der Abbildungen 2 und 3 deutlich gemacht.
Trotz der negativen Resultate aus Beispiel 4.5 und Satz 4.6 bemerken wir, dass

unseren Experimenten zufolge nichtzusammenhängende Pareto-Graphen sehr selten sind oder in unserem Fall möglicherweise gar nicht existieren.
Wir schließen daraus, dass Neighborhood Search Algorithmen und Approximationen der effizienten Menge, wie zum Beispiel die *Adjacent Search* von Andersen u.a. [1], oder verschiedene Verfahren speziell für das bikriterielle MSTP von Chen [5] oder Steiner und Radzik [21] sehr gute Ergebnisse liefern. In den meisten Fällen wird man alle effizienten spannenden Bäume finden und nur in Ausnahmefällen werden Einzelne fehlen. Denn bei den bisher vorgestellten Gegenbeispielen sind immer nur einzelne Knoten des zugehörigen Pareto-Graphen isoliert und der Rest zusammenhängend.
Bezüglich weiterer Forschungen auf diesem Gebiet gibt es drei Hauptrichtungen, die man verfolgen sollte. Zum einen besteht die Möglichkeit, den korrigierten Algorithmus 3 weiter zu analysieren und zu verbessern. In den Iterationen werden zum einen doppelte Teilgraphen berechnet und zum anderen sehr viele Lösungen, die später dominiert werden. Der erste Ansatz wäre hier zu überprüfen, ob man den Dominanzcheck des ursprünglichen Algorithmus 2 in Schritt 4 nicht anpassen könnte. Zusätzlich kann das implementierte Programm durch effizientere Methoden zum Berechnen der Schnitte und Teilgraphen sowie durch intelligentere Datenstrukturen, wie zum Beispiel Heaps, verbessert werden.
Eine andere Vorgehensweise wäre, alternative Algorithmen, wie zum Beispiel den Algorithmus von Kruskal [18], in entsprechenden multikriteriellen Versionen zu implementieren. Ehrgott [8] stellte dafür beispielsweise einen *Greedy Algorithm* vor, welcher auf einer topologischen Ordnung basiert.
Als letztes bleibt die Möglichkeit zu beweisen oder zu widerlegen, dass bei dem bikriteriellen Minimum Spanning Tree Problem mit Kosten $\{0, 1, 2\}$ in der zweiten Komponente die effiziente Menge zusammenhängend ist. Hierbei ist es vielleicht möglich die Struktur der Pareto-Graphen näher zu untersuchen und möglicherweise Gegenbeispiele zu konstruieren. Als Ansatz bieten sich Pareto-Graphen an, bei welchen ein Knoten v existiert, der nur zu einer Kante inzident ist. Der zugehörige effiziente spannende Baum unterscheidet sich damit nur in einer Kante von einem anderen EST. Diesen anderen Baum könnte man nun durch Veränderungen der Kantengewichte so modifizieren, dass er dominiert wird. Damit wäre der Knoten v isoliert und die effiziente Menge nicht mehr zusammenhängend.

Anhang

Verschiedene zufällig erzeugte Graphen mit zugehörigem Pareto-Graph für unterschiedliche Knotenanzahlen n, Kantenanzahlen m und das Maximalgewicht 100. Die Knoten der Pareto-Graphen tragen die Nummer des zugehörigen erzeugten Teilbaums beim Algorithmus.

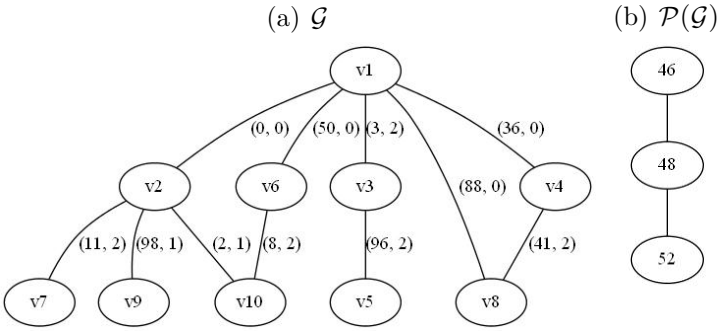


Abbildung 4: $n = 10$, wenig Kanten

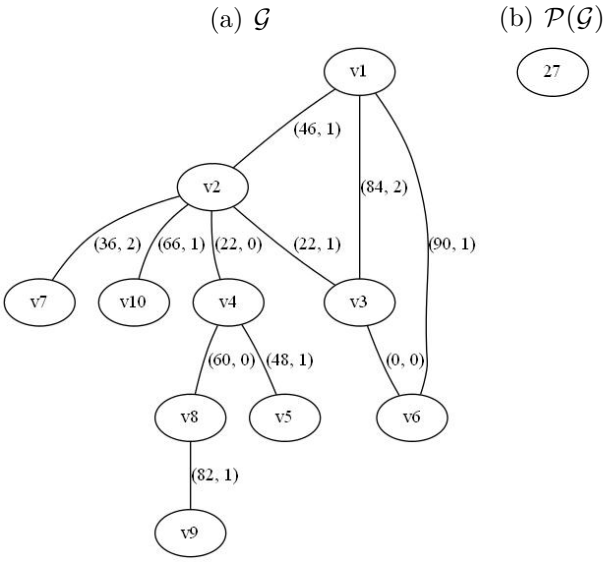


Abbildung 5: $n = 10$, wenig Kanten

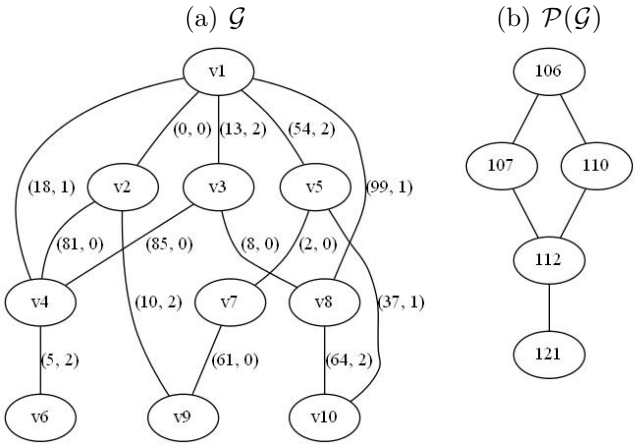


Abbildung 6: $n = 10$, mittel viele Kanten

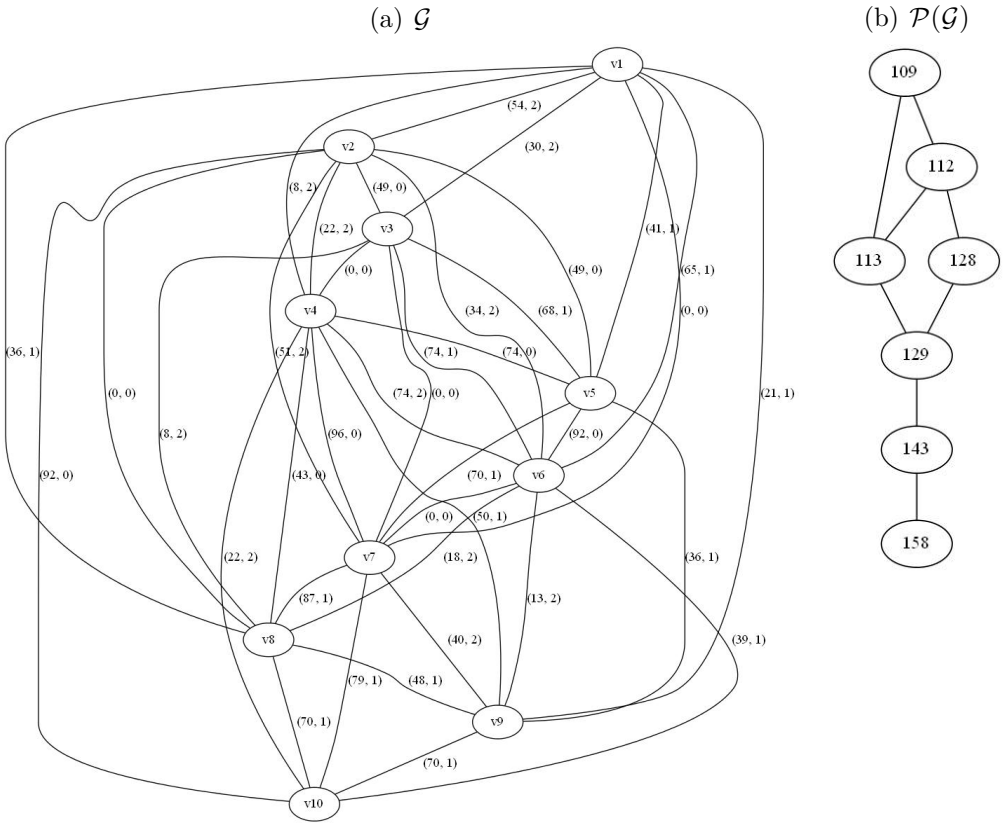


Abbildung 7: $n = 10$, viele Kanten

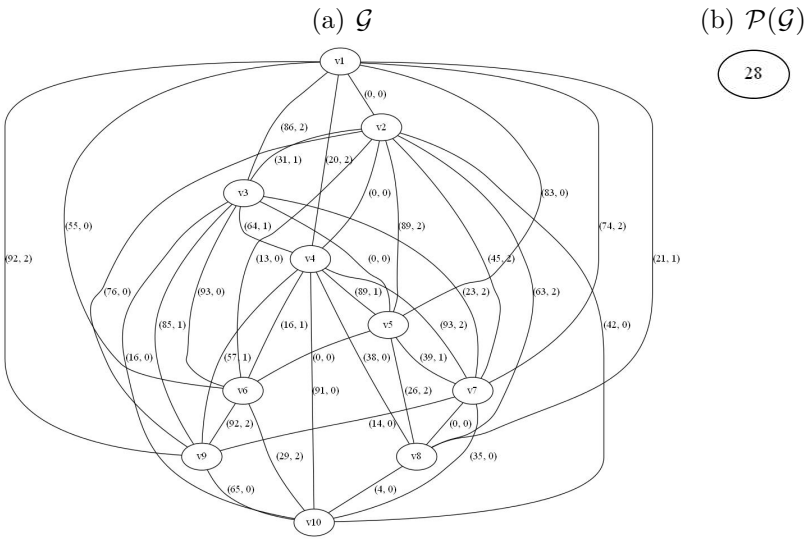


Abbildung 8: $n = 10$, viele Kanten

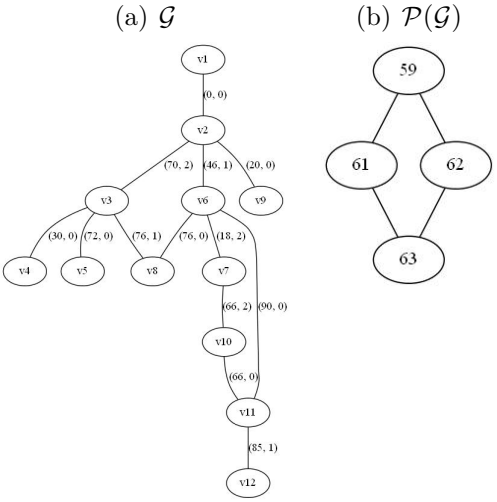


Abbildung 9: $n = 12$, wenig Kanten

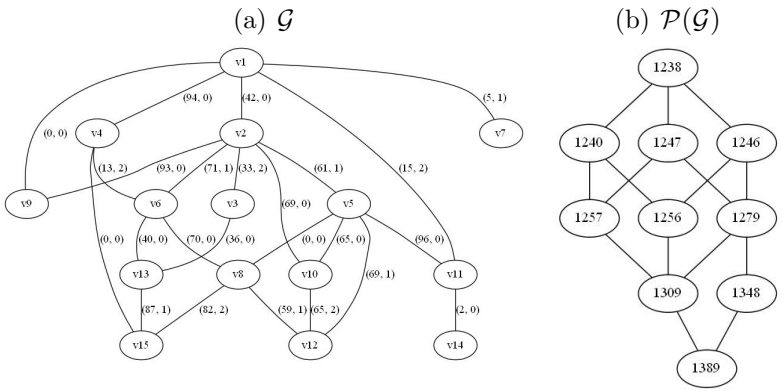


Abbildung 10: $n = 15$, mittel viele Kanten

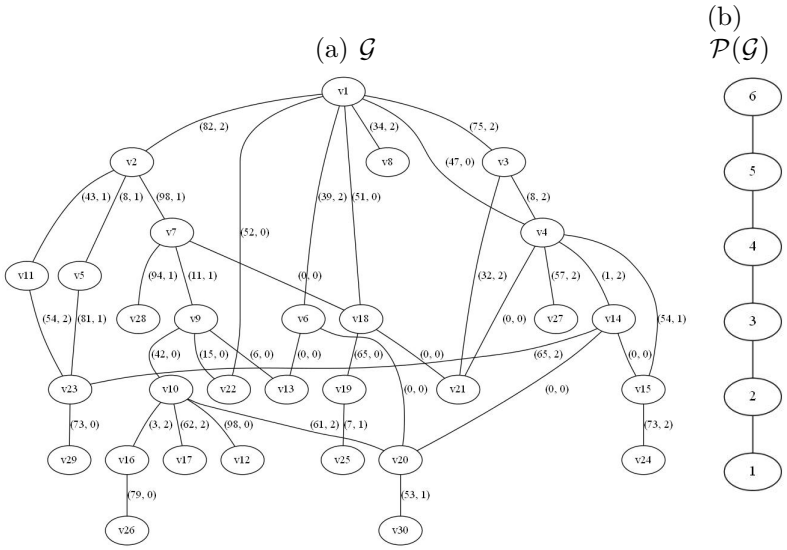
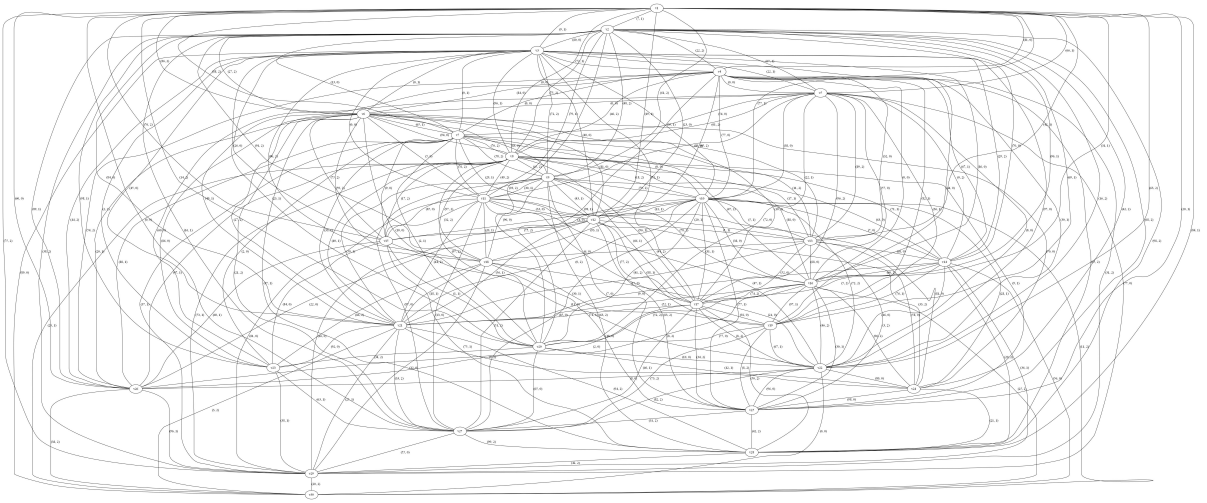


Abbildung 11: $n = 30$, wenig bis mittel viele Kanten, durchnummerierte Knoten beim Pareto-Graphen

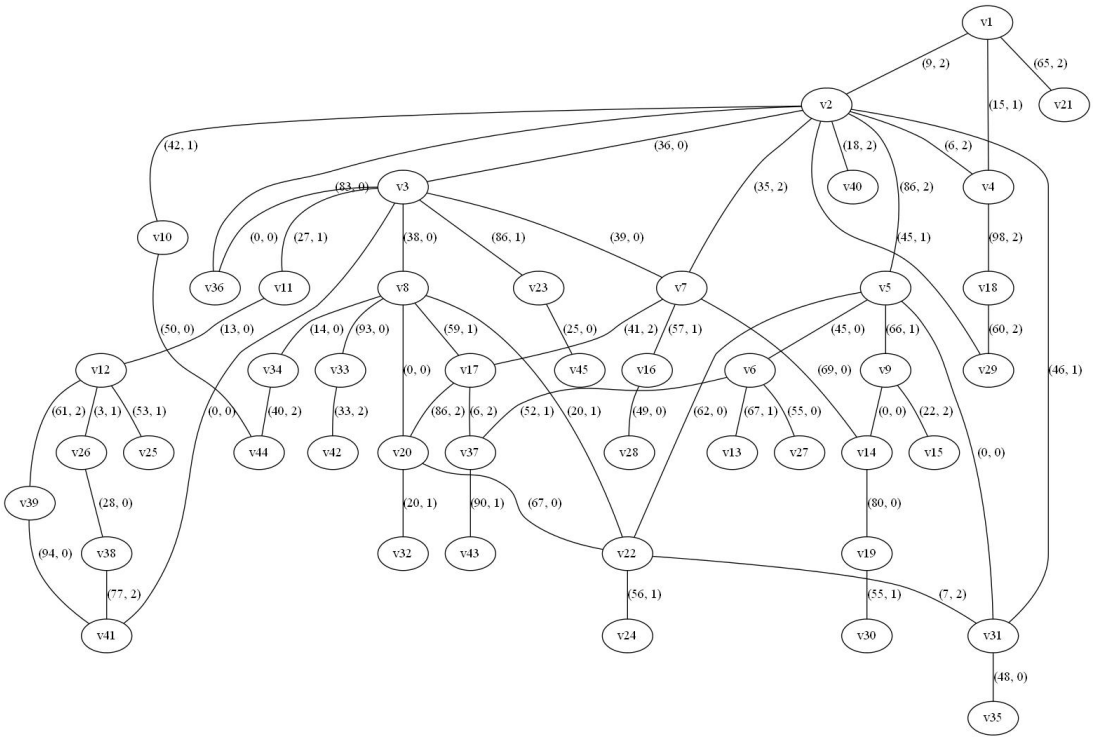
(a) $\mathcal{G}$



(b) $\mathcal{P}(\mathcal{G})$

Abbildung 12: $n = 30$, sehr viele Kanten

(a) $\mathcal{G}$



(b) $\mathcal{P}(\mathcal{G})$

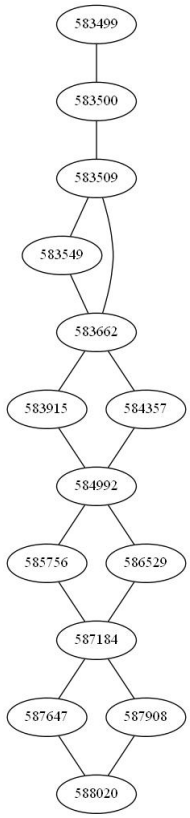
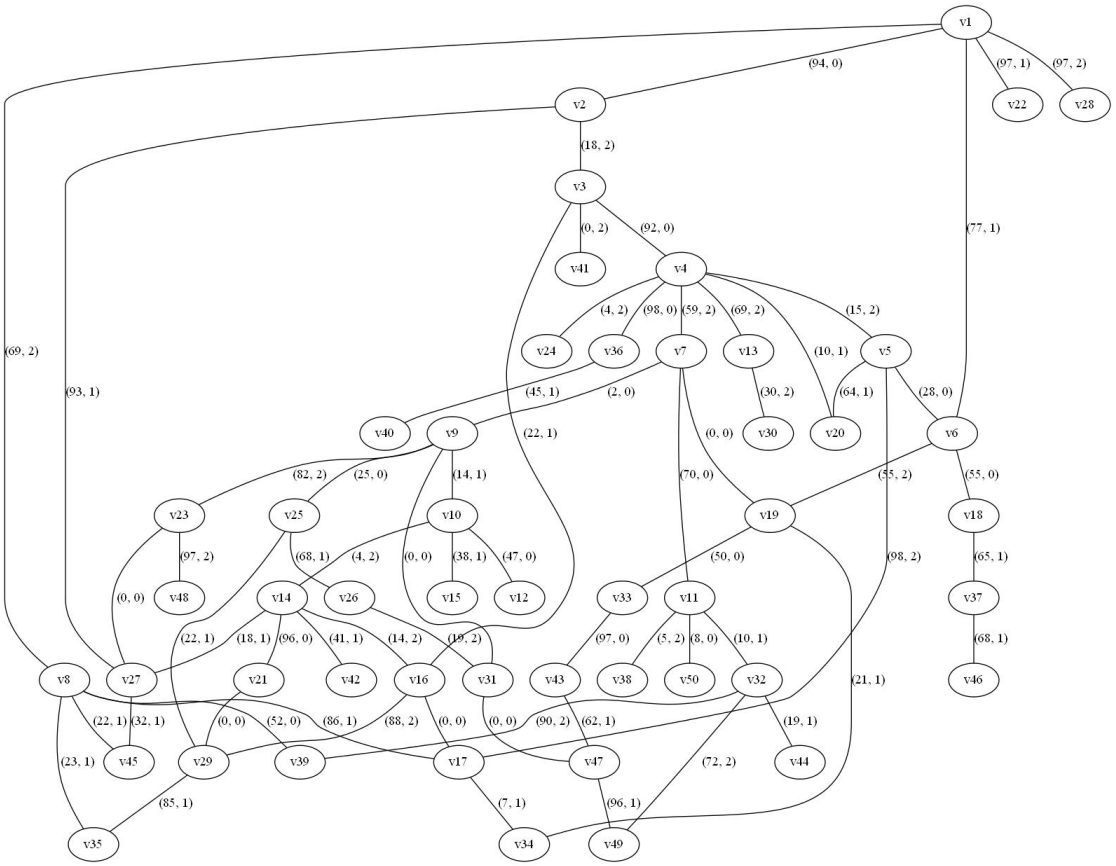


Abbildung 13: $n = 45$, wenig Kanten

(a) $\mathcal{G}$



(b) $\mathcal{P}(\mathcal{G})$

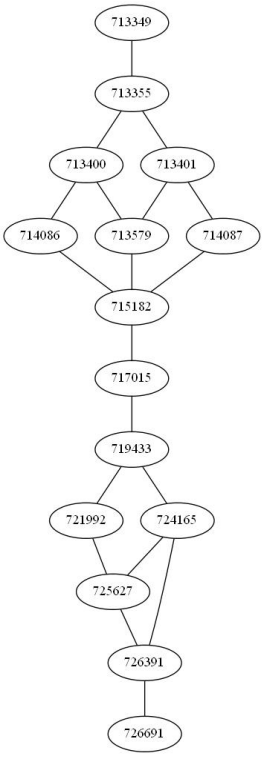
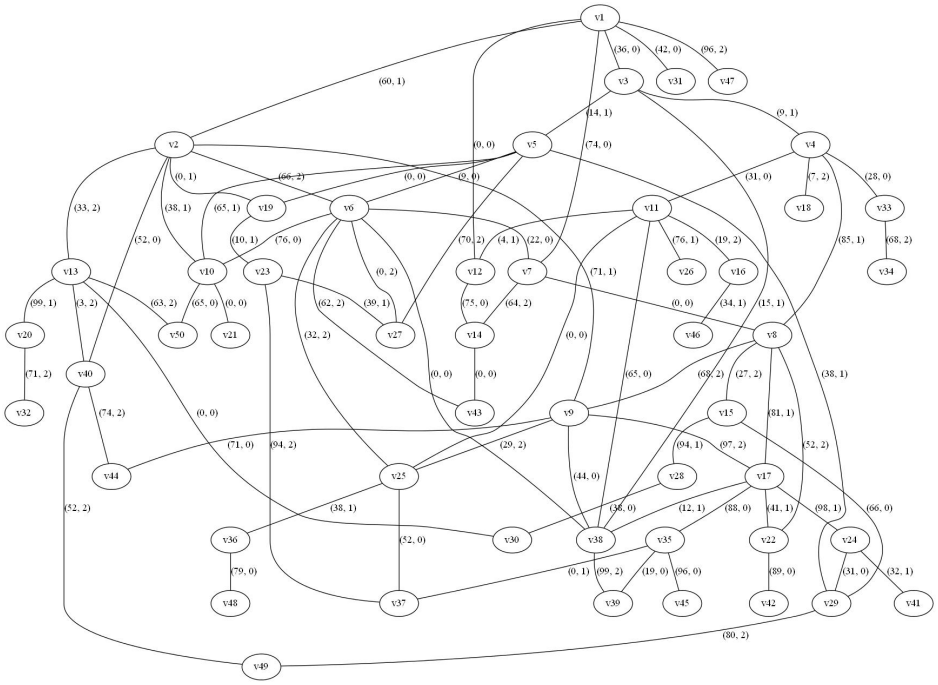


Abbildung 14: $n = 50$, wenig Kanten

(a) $\mathcal{G}$



(b) $\mathcal{P}(\mathcal{G})$

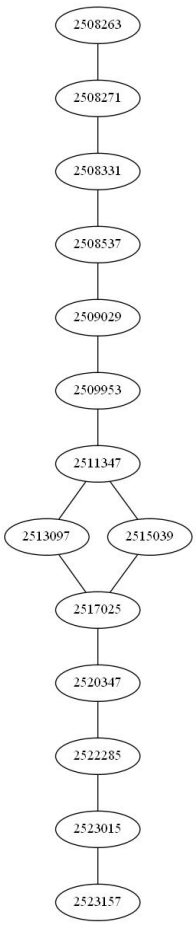


Abbildung 15: $n = 50$, wenig bis mittel viele Kanten

Beobachtete Werte (Minima, Maxima und Mittelwerte) von $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ für jeweils 100 zufällig erzeugte Graphen für $n = 10, \ldots, 35$, ein Maximalgewicht von 100 und wenig Kanten ($m = \frac{n}{2}$, $nullen = \frac{n}{6}$):

n	Minimum	Mittelwert	Maximum
10	9	68	349
11	10	83	391
12	15	119	418
13	28	174	769
14	23	235	1590
15	24	355	1819
16	45	479	2623
17	54	645	3690
18	77	972	5020
19	96	1217	9047
20	115	1762	23489
21	160	2972	21096
22	122	3457	29319
23	443	4593	53521
24	256	6506	62263
25	354	7256	68015
26	668	10146	75849
27	655	11673	101484
28	638	16612	94111
29	494	19866	198941
30	1010	30725	271176
31	1948	32495	258648
32	1469	42172	299438
33	1626	46336	403806
34	2752	97732	1748170
35	3939	127365	1222689

Tabelle 2: Minimum, Mittelwert, Maximum von $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ in Abhängigkeit von n bei jeweils 100 zufällig erzeugten Graphen mit $m = n/2$, $nullen = n/6$

Beobachtete Werte (Minima, Maxima und Mittelwerte) von $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ für jeweils 100 zufällig erzeugte Graphen für $n = 10, \ldots, 25$, ein Maximalgewicht von 100 und mittel viele Kanten ($m = \frac{n \cdot n}{2}$, $nullen = \frac{n}{4}$):

n	Minimum	Mittelwert	Maximum
10	23	176	689
11	36	302	2018
12	46	339	1611
13	35	553	3236
14	78	1278	8204
15	76	2050	15313
16	129	2077	18793
17	178	3090	22924
18	237	5928	74418
19	243	7939	48945
20	196	8255	50912
21	404	16934	230655
22	417	22011	291034
23	1025	47214	921021
24	1905	52041	477286
25	2683	102172	1032649

Tabelle 3: Minimum, Mittelwert, Maximum von $\sum_{k=1}^{n-1} |\mathcal{T}_k|$ in Abhängigkeit von n bei jeweils 100 zufällig erzeugten Graphen mit $m = {}^{n \cdot n}/_{2}$, $nullen = {}^{n}/_{4}$

Literatur

[1] Andersen, Kim A. ; Jörnsten, Kurt ; Lind, Mikael: On Bicriterion Minimal Spanning Trees: An Approximation. In: *Computers & operations research* 23 (1996), Nr. 12, S. 1171–1182

[2] Camerini, P.M. ; Galbiati, G. ; Maffioli, F.: On the complexity of finding multi-constrained spanning trees. In: *Discrete Applied Mathematics* 5 (1983), Nr. 1, S. 39–50

[3] Cayley, Arthur: A theorem on trees. In: *Quarterly Journal of Pure and Applied Mathematics* 23 (1889), S. 376–378

[4] Chazelle, Bernard: A minimum spanning tree algorithm with inverse-Ackermann type complexity. In: *Journal of the Association for Computing Machinery* 47 (2000), Nr. 6, S. 1028–1047

[5] Chen, Yen H.: Polynomial time approximation schemes for the constrained minimum spanning tree problem. In: *Department of Computer Science* (2012)

[6] Corley, H.W.: Efficient Spanning Trees. In: *Journal of Optimization Theory and Applications* 45 (1997), Nr. 3, S. 481–485

[7] Ehrgott, Matthias: On matroids with multiple objectives. In: *Optimization* 38 (1996), Nr. 1, S. 73–84

[8] Ehrgott, Matthias: *Multicriteria optimization.* 2. Auflage. Basel : Springer Verlag, 2005. – ISBN 3642059759

[9] Ehrgott, Matthias ; Klamroth, Kathrin: Connectedness of efficient solutions in multiple criteria combinatorial optimization. In: *European journal of operational research* 97 (1997), Nr. 1, S. 159–166

[10] Gabow, H. N. ; Galil, Z. ; Spencer, T. ; Tarjan, R. E.: Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. In: *Combinatorica* 6 (1986), Nr. 2, S. 109–122

[11] Gorski, Jochen: *Multiple Objective Optimization and Implications for Single Objective Optimization*, Bergische Universität Wuppertal, Dissertation, 2010

[12] Gorski, Jochen ; Klamroth, Kathrin ; Ruzika, Stefan: Connectedness of Efficient Solutions in Multiple Objective Combinatorial Optimization. In: *Journal of Optimization Theory and Applications* 150 (2011), S. 475–497

[13] Hamacher, Horst W. ; Klamroth, Kathrin: *Lineare Optimierung und Netzwerkoptimierung*. 2. Auflage. Vieweg+Teubner, 2006. – ISBN 3834801852

[14] Hamacher, Horst W. ; Ruhe, Günter: On spanning tree problems with multiple objectives. In: *Annals of Operations Research* 52 (1994), Nr. 4, S. 209–230

[15] Jarnik, Vojtech: O jistem problemu minimalnim (Über ein Minimalproblem). In: *Prace Moravske prirodovedecke spolecnosti* 6 (1930), S. 57–63

[16] Korte, Bernhard ; Vygen, Jens: *Combinatorial optimization - theory and algorithms*. 4. Auflage. Berlin : Springer Verlag, 2008. – ISBN 3540718435

[17] Krumke, Sven O. ; Noltemeier, Hartmut: *Graphentheoretische Konzepte und Algorithmen*. 2. Auflage. Vieweg+Teubner, 2009. – ISBN 3834806293

[18] Kruskal, Joseph B.: On the shortest spanning subtree of a graph and the traveling salesman problem. In: *Proceedings of the American Mathematical Society* 7 (1956), Nr. 1, S. 48–50

[19] Prim, Robert C.: Shortest Connection Networks And Some Generalizations. In: *The Bell System Technical Journal* 36 (1957), Nr. 6, S. 1389–1401

[20] Shioura, Akiyoshi ; Tamura, Akihisa ; Uno, Takeaki: An optimal Algorithm for Scanning all Spanning Trees of Undirected Graphs. In: *SIAM journal on computing* 26 (1997), Nr. 3, S. 678–692

[21] Steiner, Sarah ; Radzik, Tomasz: Solving the Biobjective Minimum Spanning Tree problem using a k-best algorithm / Department of Computer Science King's College London. 2003. – Forschungsbericht

[22] Westphal, Jun.-Prof. Dr. S.: *Vorlesung Netzwerkflüsse*. Georg-August-Universität Göttingen. WS 2010/11

Benutze Programme

- Graph File Editor *GVEdit* v1.01 von *Graphviz* v2.28
- Dev-C++ Version v4.9.9.2
- Cygwin v1.7.9-1
- MiKTeX v2.9
- Texmaker v3.2.2
- Tinn-R v2.3.7.1
- R v2.13.0

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe.

Göttingen, den 09. Februar 2012