Universität Bremen

Fachbereich 3: Mathematik und Informatik

Bachelorarbeit

Ausgewählte Algorithmen für das Steinerbaumproblem

Autor Tom-Eric Lehmkuhl

7. September 2021

1. Gutachterin: Dr. Sabine Kuske
2. Gutachter: Prof. Dr. Hans-Jörg Kreowski

Inhaltsverzeichnis

1 Einleitung

Das Steinerbaumproblem gehört zu den NP-Vollständigen Problemen der Graphentheorie. Es geht darum eine ausgewählte Menge an Knoten aus einem Graphen zu einem Baum mit minimalen Gesamtkosten zu verbinden. Dazu dürfen beliebig weitere Knoten aus dem Graphen benutzt werden. Das Steinerbaumproblem hat viele praktische Anwendungsfälle, wie zum Beispiel in Streckennetzen. Klassische Beispiele hierfür sind Autobahn- oder Schienennetze, bei denen das kleinste Netz zwischen einer Auswahl an Städten gefunden werden soll. So könnte beispielsweise das kürzeste Netz zwischen allen Großstädten gesucht werden, um dieses zu modernisieren oder auszubauen. Auch in Kommunikationsnetzen findet es Anwendung. Zum Beispiel wenn eine Gruppe an Teilnehmern miteinander kommunizieren möchte und dafür aus einem Gesamtnetz das Teilnetz mit der kürzesten Übertragungszeit sucht, das alle Teilnehmer verbindet. [Röh08] [KV08]

Beim Entwurf von Schaltungen gibt es die Steinerbaumverdrahtung bei der Steinerbäume benutzt werden, um die Verbindung der Netzsegmente zu optimieren [Lie16]. Auch bei Pipelines sind Steinerbäume gefragt, um Rohrleitungen zu sparen, wie es in einem Spiel der TU München der Fall ist [Mün15]. Grundsätzlich kann das Steinerbaumproblem also in Netzen aller Art Anwendung finden, wenn es darum geht kürzeste bzw. sparsamste Teilnetze zu finden.

Als lokales Beispiel könnte man Bremens Fahrradnetz betrachten. Davon ist ein Ausschnitt in Abbildung 1.1 dargestellt, wobei die blauen Straßen die Fahrradwege sind. Betrachtet man die Kreuzungen als Knoten und die Straßenabschnitte dazwischen als Kanten, so liegt ein Graph vor. Um die Fahrradwege besser auszubauen und zusätzliche Fahrradstraßen einzurichten, könnte ein minimaler Steinerbaum zwischen den wichtigsten Orten errechnet werden. Diese Orte könnten Stadtteile, Bildungseinrichtungen, wie die Universität/Hochschule Bremen, Bahnhöfe und vieles mehr sein. Dadurch wäre es möglich, mit dem Ausbau einer möglichst kleinen Gesamtstrecke möglichst viele Orte zu erreichen oder ein entsprechendes Netz aus Fahrradstraßen zwischen diesen Orten zu planen. Durch die geringe Gesamtstrecke würden die Kosten dabei möglichst gering gehalten werden.

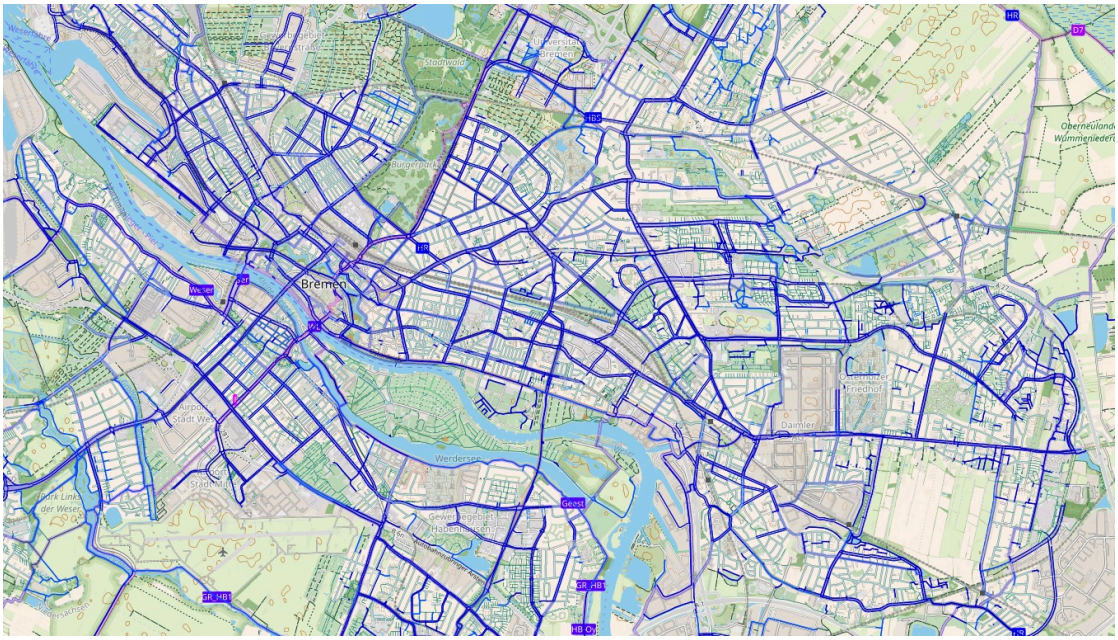


Abbildung 1.1 Ausschnitt aus dem Fahrradnetz von Bremen aus [Cyc21]

Für das Steinerbaumproblem gibt es eine Vielzahl an Algorithmen. Die exakten Algorithmen laufen dabei natürlich nicht in Polynomialzeit. Daher gibt es auch eine Reihe von Approximationsalgorithmen, die eine gute, aber nicht unbedingt optimale Lösung ausgeben. Diese deutlich schnelleren Algorithmen laufen dafür in polynomieller Zeit und es wurde versucht, diese im Laufe der Zeit immer weiter zu optimieren. [KV08, S. 535]

Diese Arbeit beschäftigt sich mit dem Steinerbaumproblem und zwei der bekanntesten Algorithmen. Ziel ist es, die in der Literatur oft nur knapp beschriebenen Algorithmen und zugehörige Beweise ausführlich und mit Skizzen und Beispielen unterstützt nachvollziehbar aufzubereiten. Damit soll diese Arbeit einen einfachen Einstieg in das Steinerbaumproblem und den Algorithmen geben, damit Leser, wie interessierte Studenten, problemlos in das Thema finden und die Inhalte auch in der Lehre benutzt werden können. Vorausgesetzt werden nur Grundkenntnisse in der Mathematik und in der theoretischen Informatik, welche in den Pflichtmodulen vermittelt werden. Daher kann diese Arbeit gut dazu benutzt werden, um das Steinerbaumproblem und die vorgestellten Algorithmen in einem Wahlmodul zu behandeln.

Die vorgestellten Algorithmen sind der Approximationsalgorithmus von Kou, Markowsky und Berman [KMB81] und der exakte Algorithmus von Dreyfus und Wagner [DW71]. Damit behandelt die Arbeit einen effizienten Algorithmus, mit dem sich eine gute, aber

nicht unbedingt optimale Lösung finden lässt und einen exakten Algorithmen mit exponentieller Laufzeit. Die Algorithmen wurden ausgewählt, da sie zu den bekannteren Lösungsalgorithmen für das Steinerbaumproblem gehören und der Kou-Markowsky-Berman-Algorithmus in seiner Funktionsweise einfach zu verstehen ist. Der Dreyfus-Wagner-Algorithmus hingegen ist komplexer, liefert aber eben auch eine optimale Lösung.

Die Arbeit beginnt mit einer Einführung in die benötigten Grundlagen zu Graphen und Bäumen. Anschließend werden Steinerbäume definiert und das Steinerbaumproblem vorgestellt. Als erster Algorithmus wird der Kou-Markowsky-Berman-Algorithmus behandelt und als zweites der Dreyfus-Wagner-Algorithmus. Bei beiden wird neben der Funktionsweise auch die Korrektheit gezeigt und die Laufzeit bestimmt. Abschließend folgt eine Zusammenfassung und ein Ausblick auf weitere Algorithmen für das Steinerbaumproblem und mögliche Anknüpfungspunkte an diese Arbeit.

2 Graphen und Bäume

In diesem Kapitel werden die Grundlagen vorgestellt, welche für eine genauere Betrachtung des Steinerbaumproblems und der Algorithmen benötigt werden. Dazu werden zunächst Graphen und einige ihrer Eigenschaften eingeführt. Die Arbeit bezieht sich auf endliche, ungerichtete Graphen ohne Schleifen. Es können also alle Kanten in jeder Richtung durchlaufen werden und eine Kante verbindet immer zwei unterschiedliche Knoten. Anschließend werden Bäume, welche Graphen mit speziellen Eingenschaften sind, behandelt. Als Grundlage für dieses Kapitel dienen [KV08], [Kus20] und [Die06].

2.1 Graphen

Ein ungerichteter Graph G ist ein Tupel $G = (V, E)$, wobei V eine nichtleere Menge von Knoten ist und E mit $E \subseteq \binom{V}{2}$ die Menge der Kanten. Jede Kante ist also eine Menge bestehend aus zwei verschiedenen Knoten aus der Knotenmenge V. Diese Definition entspricht einem ungerichteten Graphen. Dass heißt die Kanten können in beliebiger Richtung durchlaufen werden. Die Knoten eines Graphen werden auch als $V(G)$ und die Kanten als $E(G)$ bezeichnet. Graphen können visualisiert werden, indem die Knoten als Kreise und die Kanten als Verbindungslinien zwischen den Knoten dargestellt werden. In Abbildung 2.1 ist beispielhaft der Graph $G_{Example} = (V, E)$ mit

$$V = \{v_1, ..., v_8\}$$
$$E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_1, v_5\}, \{v_2, v_6\}, \{v_2, v_5\}, \{v_5, v_6\}, \{v_6, v_7\}, \{v_6, v_8\}, \{v_7, v_9\}\}$$

dargestellt, wobei die Knoten mit ihren Namen beschriftet sind. Wie im Beispiel ersichtlich, muss nicht jeder Knoten Teil einer Kante sein. Der Graph $G_{Example}$ wird im weiterem Verlauf des Kapitels immer wieder als Beispiel verwendet.

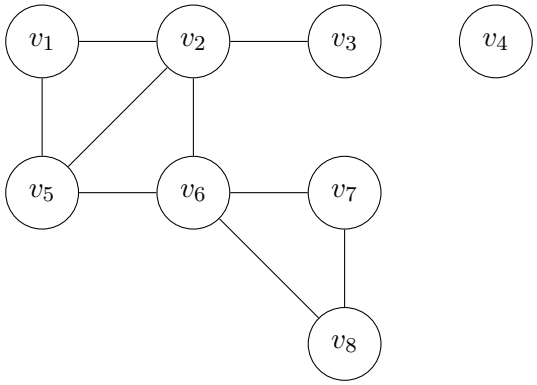


Abbildung 2.1 Graph $G_{Example}$

Der Grad eines Knoten v ist die Anzahl der Kanten, durch welche dieser direkt mit anderen Knoten verbunden ist, also $|\{e \in E \mid v \in e\}|$. Der Knoten v_6 hat zum Beispiel den Grad 4, während der Knoten v_4 den Grad 0 hat.
Zwei Knoten v und v' sind adjazent, wenn es eine Kante $e = \{v, v'\}$ im Graphen gibt. In $G_{Example}$ sind beispielsweise die Knoten v_1 und v_2 adjazent.
Die Vereinigung zweier Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ ist die Vereinigung der Knoten und Kanten, also $G_U = (V_U, E_U)$ mit

$$V_U = V_1 \cup V_2 \text{ und}$$
$$E_U = E_1 \cup E_2$$

2.1.1 Teilgraphen

Ein Teilgraph besteht, wie der Name schon schließen lässt, aus Knoten und Kanten, die Teil eines anderen Graphen sind. Ein Graph $G' = (V', E')$ ist also ein Teilgraph von G (in Zeichen $G' \subseteq G$), wenn gilt $V(G') \subseteq V(G)$ und $E(G') \subseteq E(G)$. Ein Beispiel für einen Teilgraphen von $G_{Example}$ ist in Abbildung 2.2 dargestellt. Enthält ein Teilgraph alle Knoten vom ursprünglichen Graphen, also gilt $V(G') = V(G)$, dann ist G' ein aufspannender Teilgraph (Abbildung 2.3).

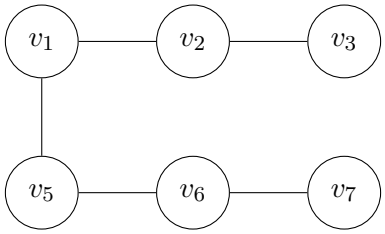


Abbildung 2.2 Ein Teilgraph $G'_{Example}$ von $G_{Example}$

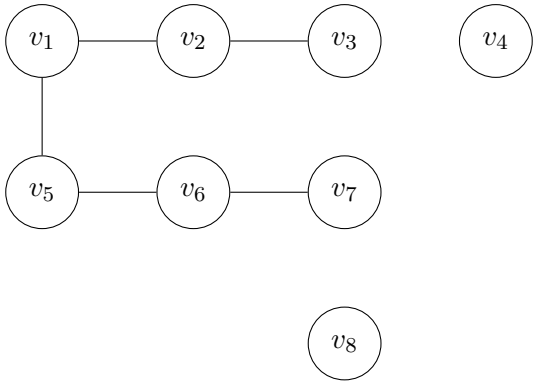


Abbildung 2.3 Ein aufspannender Teilgraph $G'_{Example}$ von $G_{Example}$

2.1.2 Wege und Kosten

Ein Weg ist eine Knotenfolge, die ohne Unterbrechung durchlaufen werden kann. Das heißt, dass zwischen je zwei aufeinanderfolgenden Knoten der Folge eine Kante im zugehörigen Graphen existieren muss. Ein Weg von v nach v' bzw. zwischen v und v' mit der Länge $n \in \mathbb{N}$ ist also eine Knotenfolge $v_0...v_n$ mit

$$v_0 = v$$
$$v_n = v'$$
$$\{v_{i-1}, v_i\} \in E \text{ mit } i = 1, ..., n$$

Ein Weg heißt nichtleer, wenn er mindestens eine Kante durchläuft. Gibt es einen Weg von v nach v', so wird v' als von v aus erreichbar bezeichnet. In Abbildung 2.4 ist ein Weg $p = v_2v_6v_8$ von v_2 nach v_8 im Graphen $G_{Example}$ in blau eingezeichnet.

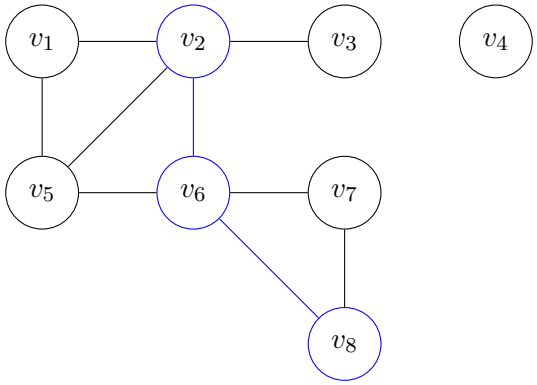


Abbildung 2.4 Weg von v_2 nach v_8 der Länge 2 in $G_{Example}$

Den Kanten eines Graphen können Kosten zugewiesen werden. Dies erfolgt über eine Kostenfunktion in Form einer Abbildung $c\colon E \to \mathbb{N}$. Den Kanten werden also natürliche Zahlen zugeordnet, die die Kosten darstellen. Es könnten natürlich auch beispielsweise reelle Zahlen als Kosten benutzt werden. In dieser Arbeit wird sich aber für eine bessere Verständlichkeit auf natürliche Zahlen beschränkt. Oftmals wird auch statt von Kosten von Distanzen gesprochen. Gibt es eine Kostenfunktion zu einem Graphen G, so wird das Tupel $G = (V, E)$ des öfteren auch zu einem Tripel $G = (V, E, c)$ erweitert. Wird ein Graph als Tupel mit einer zusätzliche Kostenfunktion angegeben, hat dies aber die selbe Bedeutung. In beiden Fällen wird, wenn vom Graphen gesprochen wird, auch immer die Kostenfunktion mit einbezogen. In Abbildung 2.5 sind Kosten im Graphen $G_{Example}$ eingetragen.

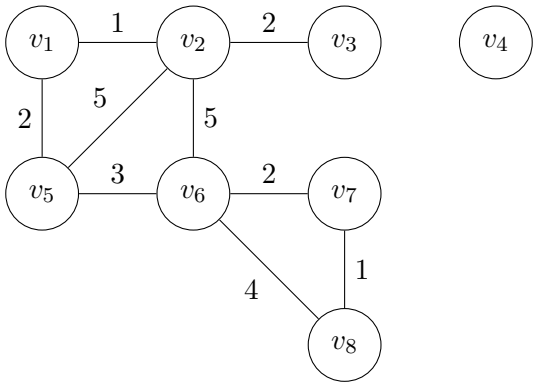


Abbildung 2.5 Graph $G_{Example}$ mit Kosten

Die Kosten eines Weges sind die Summe der Kosten der durchlaufenden Kanten. Der

Weg $p = v_2v_6v_8$ hat also in diesem Beispiel die Kosten 9. Wie leicht zu erkennen ist, gibt es einen Weg von v_2 nach v_8 mit geringeren Kosten, der über den Knoten v_7 geht. Dies ist gleichzeitig ein kürzester Weg zwischen den beiden Knoten im Graphen G.

Ein kürzester Weg zwischen zwei Knoten $v, v' \in V$ ist ein Weg mit den geringsten Kosten. Das heißt, es gibt in einem Graphen keinen anderen Weg zwischen diesen Knoten, der geringere Kosten aufweist. Die Kosten der kürzesten Wege zwischen zwei Knoten v und v' werden als $short(v, v')$ bezeichnet.

2.1.3 Kreise

Ein Kreis ist ein nichtleerer Weg der Länge $n \geq 2$, bei dem der Startknoten gleich dem Endknoten ist, je zwei aufeinanderfolgende Knoten verschieden sind und der nicht mit der gleichen Kante beginnt und endet. In der Knotenfolge gilt also $v_0 = v_n$, $v_i \neq v_{i+1}$ und $v_1 \neq v_{n-1}$. In Abbildung 2.6 ist der Kreis $v_1v_2v_6v_5v_1$ farblich hervorgehoben. Gibt es keinen Kreis in einem Graphen, so wird er als kreisfrei bezeichnet.

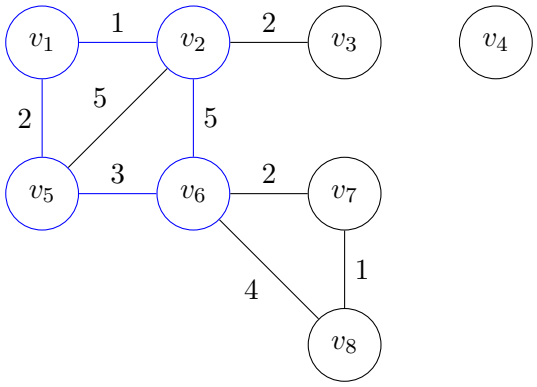


Abbildung 2.6 Kreis in $G_{Example}$

2.1.4 Zusammenhang

Ein Graph wird als zusammenhängend bezeichnet, wenn es zwischen je zwei Knoten des Graphen einen Weg gibt, also für alle $v, v' \in V$ ein Weg von v nach v' existiert. Der Graph $G_{Example}$ ist offensichtlich nicht zusammenhängend, da es von keinem anderen Knoten einen Weg zum Knoten v_4 gibt. Der Graph ohne diesen Knoten, also $G'_{Example} = (V \backslash \{v_4\}, E)$ hingegen ist zusammenhängend.

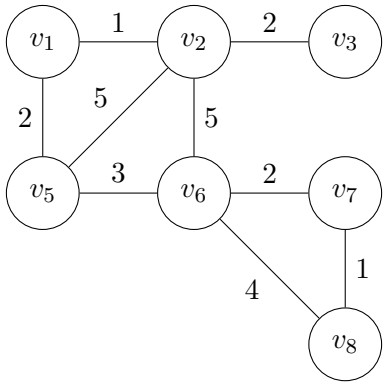


Abbildung 2.7 Zusammenenhängender Graph $G'_{Example}$

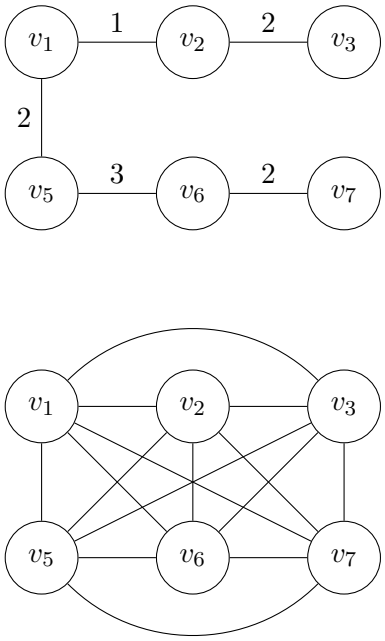
2.1.5 Metrischer Abschluss

Der metrische Abschluss eines Graphen $G = (V, E)$ mit Distanzfunktion c ist das Tupel (G_M, c_M). Dabei hat der Graph G_M die gleichen Knoten wie G. Für je zwei verschiedene Knoten $v, v' \in V$ enthält G_M genau dann eine Kante e, wenn es einen Weg von v nach v' in G gibt, wobei e die Kosten von $short(v, v')$ aus G zugeordnet werden. Für eine bessere Lesbarkeit wird im Verlauf der Arbeit der metrische Abschluss nur als G_M bezeichnet. Die zugehörige Distanzfunktion ist dabei inbegriffen. Abbildung 2.8 zeigt den Teilgraphen aus Abbildung 2.2 und den zugehörigen metrischen Abschluss. Für eine bessere Übersicht befinden sich die Kosten in einer Tabelle daneben.

Der metrische Abschluss $G_M(T)$ auf einen Graphen $G = (V, E)$ bezüglich einer Teilmenge der Knoten $T \subseteq V$ ist der Teilgraph des metrischen Abschlusses G_M von G mit

$$V(G_M(T)) = T \text{ und}$$
$$E(G_M(T)) = E(G_M) \cap \binom{T}{2}$$

In Abbildung 2.9 ist der metrische Abschluss für das Beispiel aus Abbildung 2.8 bezüglich der Knotenmenge $T = \{v_1, v_2, v_5, v_6\}$ abgebildet.



(a) Teilgraph $G_{Example}$ (oben) und dessen metrischer Abschluss (unten)

Kante	**Distanz**
$\{v_1, v_2\}$	1
$\{v_1, v_3\}$	3
$\{v_1, v_5\}$	2
$\{v_1, v_6\}$	5
$\{v_1, v_7\}$	7
$\{v_2, v_3\}$	2
$\{v_2, v_5\}$	3
$\{v_2, v_6\}$	6
$\{v_2, v_7\}$	8
$\{v_3, v_5\}$	5
$\{v_3, v_6\}$	8
$\{v_3, v_7\}$	10
$\{v_5, v_6\}$	3
$\{v_5, v_7\}$	5
$\{v_6, v_7\}$	2

(b) Kosten für metrischen Abschluss

Abbildung 2.8 Beispiel für metrischen Abschluss

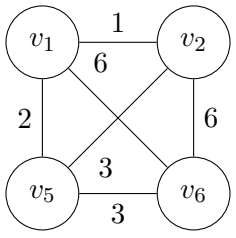


Abbildung 2.9 Metrischer Abschluss auf einer Teilmenge

2.2 Bäume

Ein Graph ist ein Baum, wenn er zusammenhängend und kreisfrei ist. Ein Graph kann mehrere Bäume als Teilgraphen enthalten. Ein Baum B, der Teilgraph eines Graphen G ist und alle Knoten $V(G)$ enthält, ist ein aufspannender Baum. Knoten mit dem Knotengrad 1 werden als Blätter bezeichnet. Alle anderen Knoten werden als innere Knoten bezeichnet. In Abbildung 2.10 sind Bäume aus dem Graph $G_{Example}$ aus Abbildung 2.5 abgebildet.

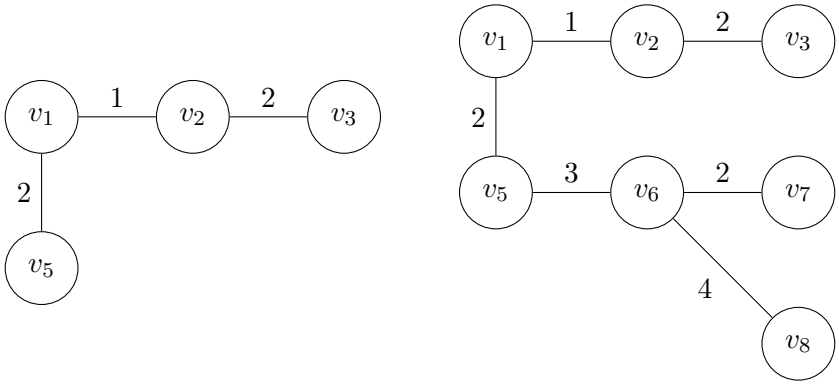
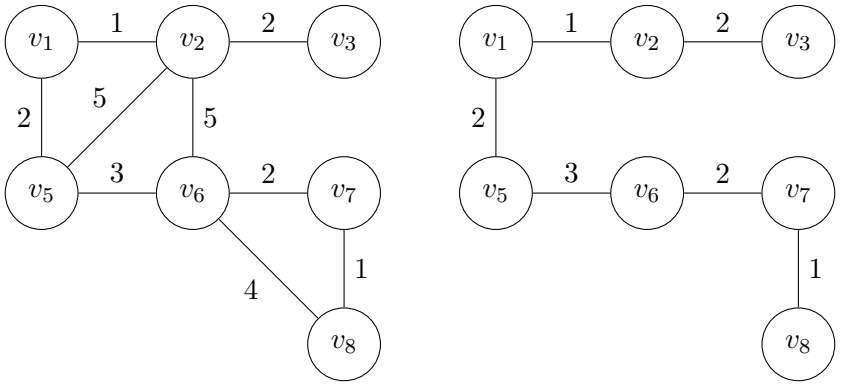


Abbildung 2.10 Bäume aus $G_{Example}$

2.2.1 Minimale aufspannende Bäume

Für einen Graph $G = (V, E)$ ist ein aufspannender Baum minimal, wenn es keinen aufspannenden Baum mit geringeren Kosten in G gibt. Die Kosten $c(B)$ für einen Baum B definieren sich über die Summe der Kosten der enthaltenen Kanten. Ein Graph kann mehrere minimale aufspannende Bäume enthalten. Abbildung 2.11 zeigt einen Graphen $G'_{Example}$ und einen zugehörigen minimalen aufspannenden Baum B.



(a) Graph $G'_{Example}$ **(b)** Baum B

Abbildung 2.11 Minimaler aufspannender Baum

3 Das Steinerbaumproblem

In diesem Kapitel wird das Steinerbaumproblem definiert und an Beispielen erklärt. Als Grundlage hiefür dienen [Röh08] und [KV08]. Außerdem wird mittels einer Reduktion von Vertex-Cover die NP-Vollständigkeit bewiesen. Zuletzt wird noch auf auf polynomielle Spezialfälle des Problems eingegangen.

3.1 Definition

Ein Teilgraph S von einem Graph $G = (V, E)$ ist ein Steinerbaum für $T \subseteq V(G)$, wenn er ein Baum ist und alle Knoten aus T enthält. Die Knoten aus T werden dabei als Terminale bezeichnet. Die Knoten der Menge $V(S) \backslash T$ heißen Steinerknoten. In manchen Definitionen wird zusätzlich noch gefordert, dass alle Blätter eines Steinerbaums Terminale sein müssen.

Beim Steinerbaumproblem geht es darum, minimale Steinerbäume zu finden. Es ist wie folgt definiert:

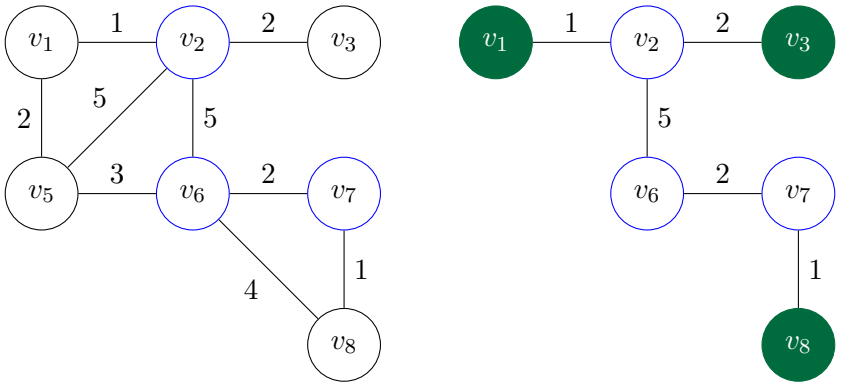
Gegeben:
Ein Graph $G = (V, E)$ mit einer Distanzfunktion $c\colon E \to \mathbb{N}$ und eine Menge $T \subseteq V$.
Gesucht:
Ein Steinerbaum S für die Terminalmenge T in G mit

$$\sum_{e \in E(S)} c(e)$$

ist minimal.

3.2 Beispiele

Im folgendem Beispiel ist für den Graphen $G_{Example}$ ein minimaler Steinerbaum S für $T = \{v_1, v_3, v_8\}$ konstruiert worden. Die Steinerknoten sind blau und die Terminale grün hervorgehoben.



(a) Graph $G_{Example}$ **(b)** Steinerbaum S

Abbildung 3.1 Minimaler Steinerbaum

Die Knoten der Menge T wurden also durch Hinzunahme der Knoten v_2, v_6 und v_7 mit einer Kantenmenge minimaler Distanz verbunden. Es ist leicht zu erkennen, dass es noch einen alternativen minimalen Steinerbaum gibt:

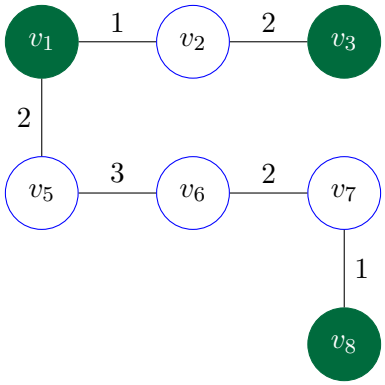


Abbildung 3.2 Weiterer minimaler Steinerbaum

3.3 NP-Vollständigkeit

Um die NP-Vollständigkeit zu zeigen, wird Vertex-Cover auf das Steinerbaumproblem reduziert [KV08]. Vertex-Cover ist ein Entscheidungsproblem, das NP-vollständig ist und als Eingabe einen Graphen $G = (V, E)$ und eine natürliche Zahl k hat. Die Fragestellung lautet: Gibt es eine Knotenüberdeckung der Kardinalität k?
Eine Knotenüberdeckung ist eine Teilmenge U der Knoten V aus einem Graphen G, sodass von jeder Kante mindestens ein Knoten in U enthalten ist. Ein Beispiel für eine Knotenüberdeckung ist in Abbildung 3.3 dargestellt.

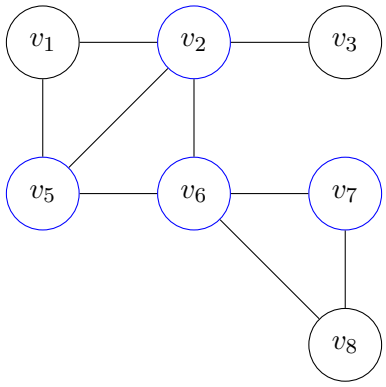


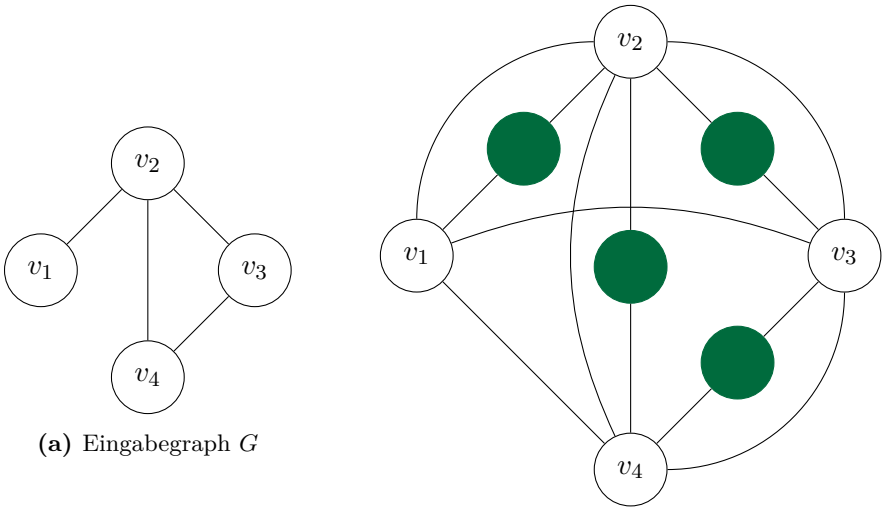
Abbildung 3.3 Knotenüberdeckung (in blau eingezeichnet)

Für die Reduktion wird das Steinerbaumproblem als Optimierungsproblem betrachtet, bei dem es darum geht den minimalen Steinerbaum für eine Eingabe zu finden. Um eine Instanz für Vertex-Cover auf das Steinerbaumproblem zu transformieren, wird aus dem Eingabegraphen G ein neuer Graph H mit $V(H) \coloneqq V(G) \cup E(G)$ erzeugt. Es gibt also für jede Kante aus G einen Knoten in $V(H)$. Die Kantenmenge ist

$$E(H) \coloneqq \binom{V(G)}{2} \cup \{\{v, e\} \mid v \in e, e \in E(G)\}\,.$$

Somit gibt es eine Kante zwischen je zwei Knoten aus G und zwischen den neu hinzugefügten Knoten und dem Knoten, die die ursprüngliche Kante aus G verbunden hat.
Als Eingabe für das Steinerbaumproblem wird der Graph H benutzt und $T \coloneqq E(G)$ und $c\colon E \to 1$ gesetzt. Als Terminale werden also die neuen Knoten benutzt und das Gewicht aller Kanten auf 1 gesetzt. In Abbildung 3.4 ist diese Transformation für einen Beispielgraphen dargestellt. Nun gilt es die Korrektheit der Reduktion zu zeigen. Dafür müssen die folgenden zwei Richtungen gezeigt werden. Anschließend wird gezeigt, dass

aus einem minimalen Steinerbaum eine minimale Knotenüberdeckung resultiert, dass die Reduktion in Polynomialzeit erfolgt und begründet, warum die NP-Vollständigkeit auch für das Steinerbaumproblem mit Kantengewichten gilt.



(a) Eingabegraph G

(b) Tranformierter Graph H mit grünen Terminalen

Abbildung 3.4 Tranformation vom Eingabegraph G zum neuen Graphen H

Die zu zeigenden Richtungen:

1. *Existiert in G eine Knotenüberdeckung $X \subseteq V(G)$, dann gibt es in H einen Steinerbaum mit X als Steinerknoten und T als Terminale.*
2. *Existiert in H ein Steinerbaum S mit T als Terminale und $X \subseteq V(G)$ als Steinerknoten, dann ist X in G eine Knotenüberdeckung.*

Zu 1:
Dadurch, dass die Knoten aus G im Graphen H vollständig miteinander vernetzt sind, existiert ein Baum mit der Knotenmenge X in H. Da X eine Knotenüberdeckung in G ist, ist in jeder Kante $e \in E(G)$ mindestens ein Knoten aus X enthalten. Nach Konstruktion gibt es also in H für jeden Knoten aus T eine Kante zu einem Knoten aus X. Dadurch lässt sich ein Steinerbaum mit X als Steinerknoten und T als Terminale in H konstruieren. Abbildung 3.5 zeigt dies beispielhaft.

Zu 2:
Da nach der Konstruktion von H Knoten aus T nur adjazent mit Knoten aus $V(G)$

sind, bedeutet die Existenz eines Steinerbaums in H mit T als Terminale und X als Steinerknoten, dass jeder Knoten aus T eine Kante zu einem Knoten aus X hat. Aufgrund der Konstruktion von H aus G besitzt daher jede Kante aus $E(G)$ mindestens einen Knoten aus X. Dies entspricht der Definition einer Knotenüberdeckung. Auch diese Richtung ist in Abbildung 3.5 zu erkennen.

Es gibt also genau dann einen Steinerbaum mit k Steinerknoten für T in H, wenn es eine Knotenüberdeckung in G mit der Kardinalität k gibt. Gibt es somit einen minimalen Steinerbaum, dann gibt es auch eine minimale Knotenüberdeckung, da bei der Transformation alle Kantengewichte auf eins gesetzt werden und das Gewicht des Steinerbaums daher nur von der Anzahl der Kanten abhängt und somit auch von der Anzahl der Knoten, da für jeden Baum B $|E(B)| = |V(B)| - 1$ gilt [Tur09, S. 58]. Da die Anzahl der Terminalknoten nicht reduzierbar ist, kann nur die Anzahl der Steinerknoten reduziert werden. Ein minimaler Steinerbaum für die Terminalmenge T in H hat also eine minimale Anzahl an Steinerknoten, woraus wiederum eine minimale Knotenüberdeckung in G folgt.

Die Reduktion erfolgt in Polynomialzeit erfolgt, da $|E(G)|$ neue Knoten und maximal $2|E(G)| + \frac{|V(G)|^2 - |V(G)|}{2}$ Kanten hinzugefügt werden. Das hinzufügen von Knoten und Kanten kann in polynomieller Zeit durchgeführt werden. Dadurch ergibt sich ein Aufwand von $\mathcal{O}(|V(G)|^2)$. Da das Steinerbaumproblem für Graphen ohne Kantengewichte ein Spezialfall für das Steinerbaumproblem für Graphen mit gewichteten Kanten ist, ist auch dieses damit NP-vollständig. □

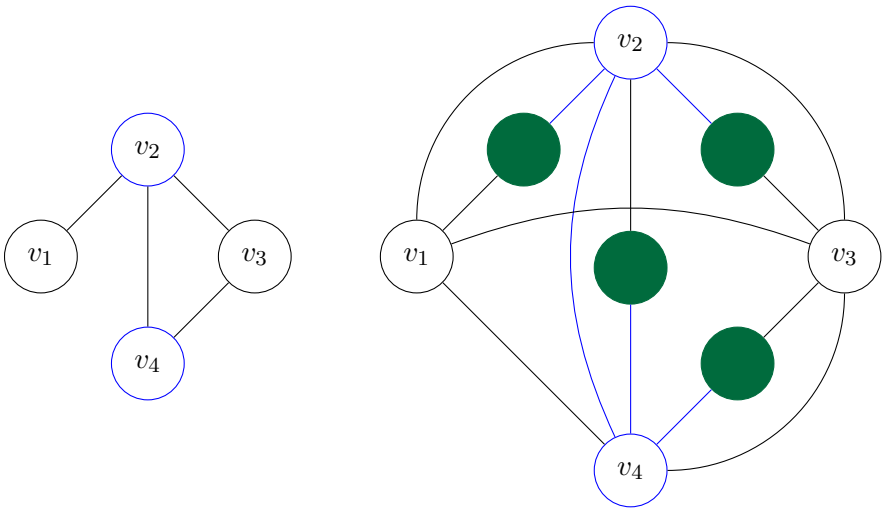


Abbildung 3.5 Beispiel für eine Knotenüberdeckung (blaue Knoten) und einem resultierenden Steinerbaum mit blauen Steinerknoten und grünen Terminalen

3.3.1 Polynomielle Spezialfälle

Für das Steinerbaumproblem gibt es einige Spezialfälle, die mit polynomiellen Aufwand gelöst werden können. Besteht zum Beispiel die Terminalmenge nur aus zwei Knoten, entspricht ein kürzester Weg einem minimalen Steinerbaum. Kürzeste Wege lassen sich mit dem Dijkstra-Algorithmus [Dij59] oder mit dem Floyd-Warshall Algorithmus [Flo62] [War62] berechnen. Noch einfacher ist es natürlich, wenn die Terminalmenge nur einen Knoten enthält, denn dann ist der Knoten selbst auch der minimale Steinerbaum. Ein weiterer Spezialfall liegt vor, wenn die Terminalmenge identisch mit der Knotenmenge des Ausgangsgraphen ist, denn dann ist ein minimaler aufspannender Baum gleichzeitig auch ein minimaler Steinerbaum für die Terminalmenge. Minimale aufspannende Bäume lassen sich beispielsweise mit dem Algorithmus von Kruskal [Kru56] oder dem Algorithmus von Prim [Pri57] berechnen.

4 Der Algorithmus von Kou, Markowsky und Berman

In diesem Teil wird der Approximations-Algorithmus von Kou, Markowsky und Berman genauer betrachtet. Dabei wird der Algorithmus an einem Beispiel erklärt und die Korrektheit bzw. die Approximationsgüte und die Laufzeit analysiert. Als Grundlage für dieses Kapitel dienen [KMB81] und [KV08], sowie [Röh08].

4.1 Funktionsweise

Der Kou-Markowsy-Berman-Algorithmus liefert nicht unbedingt einen minimalen Steinerbaum, denn er berechnet nur eine Lösung, die in der Nähe der optimalen Lösung liegt. Es kann bestimmt werden, wie nah eine Lösung des Algorithmus mindestens am Optimum liegt. Diese Kennzahl nennt sich Approximationsgüte und wird im späteren Verlauf des Kapitels untersucht.

Für jede Eingabe, bestehend aus einem zusammenhängenden Graphen $G = (V, E)$, einer Distanzfunktion $c\colon E \to N$ und einer Knotenmenge $T \subseteq V$ wird zunächst der metrische Abschluss $G_M(T)$ für die Knoten aus T berechnet. Dabei haben die Kanten zwischen den Knoten immer das Gewicht des kürzesten Weges aus G. Für $G_M(T)$ wird ein minimaler aufspannender Baum B_M konstruiert. In diesem wird anschließend jede Kante $e = \{v, v'\}$ durch einen kürzesten Weg von v nach v' aus G ersetzt, also die entsprechenden Knoten und Kanten eingefügt und damit der Graph G_B erzeugt. Für diesen wird wieder ein minimaler aufspannender Baum B_S berechnet. Durch entfernen von Wegen und Kanten, die in einem Blatt $\notin T$ beginnen bis zum ersten Terminalknoten oder einem Knoten mit einem Knotengrad größer zwei, wird aus B_S der Steinerbaum S, der ausgegeben wird.

Dadurch, dass in einem Graphen verschiedene minimale Bäume und kürzeste Wege existieren können, kann der Algorithmus bei gleicher Eingabe verschiedene Lösungen ausgeben, da, je nachdem welche Wege und Bäume verwendet werden, der resultierende Steinerbaum beeinflusst wird. Es handelt sich somit um einen nichtdeterministischen Algorithmus.

Die einzelnen Schritte nochmal kurz zusammengefasst:

1. Berechnen des metrischen Abschlusses $G_M(T)$.
2. Berechnen eines minimalen aufspannenden Baums B_M für $G_M(T)$.
3. Ersetzen der Kanten von B_M durch kürzeste Wege aus G.
4. Für den erzeugten Graphen einen minimalen aufspannenden Baum B_S konstruieren.
5. Überflüssige Wege und Kanten entfernen.

4.2 Der Algorithmus am Beispiel

Der Algorithmus wird nun Schritt für Schritt mit einem Beispielgraphen durchgespielt. Als Eingabe sei der Graph $G_{Example}$ mit gewichteten Kanten, wie in Abbildung 4.1 dargestellt, gegeben und $T = \{v_1, v_5, v_8, v_{10}\}$.

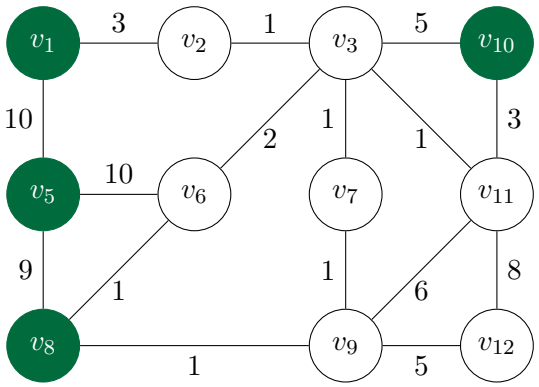


Abbildung 4.1 Eingabegraph $G_{Example}$ mit grün makierter Knotenmenge T

Für den metrischen Abschluss auf den Knoten T werden die Distanzen der kürzesten Wege zwischen den Knoten bestimmt. Dies kann zum Beispiel mit dem Algorithmus von Floyd und Warshall [Flo62], [War62] geschehen:

Knoten		**Distanz**
v_1	v_5	10
v_1	v_8	7
v_1	v_{10}	8
v_5	v_8	9
v_5	v_{10}	16
v_8	v_{10}	7

Tabelle 4.1 Distanzen der kürzesten Wege

Daraus resultiert der metrische Abschluss:

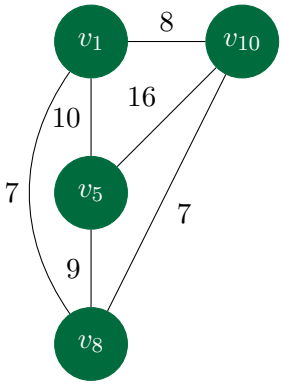


Abbildung 4.2 Metrischer Abschluss $G_M(T)$

Nun wird ein minimaler aufspannender Baum B_M aus $G_M(T)$ konstruiert. Dies ist zum Beispiel mit dem Algorithmus von Prim möglich [Pri57]:

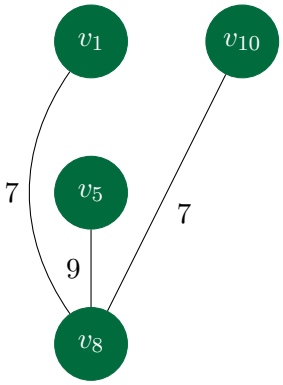


Abbildung 4.3 Minimaler aufspannender Baum B_M

Im nächsten Schritt werden die Kanten aus B_M durch kürzeste Wege aus G ersetzt. Die Kante $\{v_1, v_8\}$ wird also durch den zugehörigen Weg $v_1v_2v_3v_6v_8$ ersetzt und die Kante $\{v_8, v_{10}\}$ durch den Weg $v_8v_9v_7v_3v_{11}v_{10}$. Die Kante $\{v_8, v_5\}$ entspricht bereits dem kürzesten Weg in G. Wie zu erkennen ist, können durch das Einfügen der Wege Kreise entstehen:

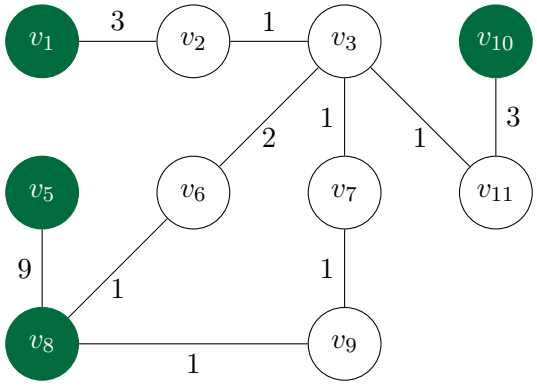


Abbildung 4.4 B_M mit kürzesten Wegen aus G

Nun wird aus diesem Graphen ein minimaler aufspannender Baum B_S konstruiert:

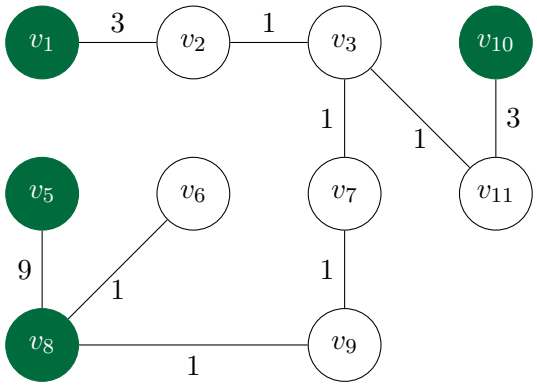


Abbildung 4.5 Baum B_S

Aus B_S werden anschließend die überflüssigen Wege und Kanten entfernt, beginnend in einem Blatt, das nicht aus T ist, bis zum ersten Knoten der aus T ist und/oder einen Knotengrad größer zwei hat. In diesem Fall ist das der leere Weg $\{v_6\}$ und die Kante $\{v_6, v_8\}$. Dadurch erhalten wir unseren Steinerbaum S:

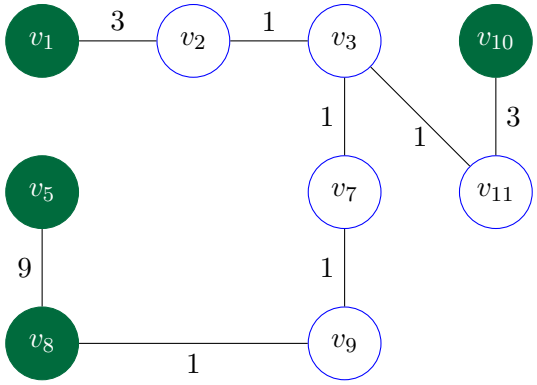


Abbildung 4.6 Steinerbaum S mit blauen Steinerknoten

4.3 Korrektheit

Zunächst wird gezeigt, dass der Algorithmus für jede Eingabe terminiert und anschließend, dass bei Termination das Ergebnis korrekt ist. Da vorausgesetzt wird, dass der Eingabegraph zusammenhängend und endlich ist, ist auch der metrische Abschluss bezüglich der Terminalmenge zusammenhängend. Um diesen zu bestimmen werden die kürzesten Wege berechnet. Dies kann zum Beispiel mit dem Floyd-Warshall-Algorithmus ge-

macht werden, welcher bei einem endlichen Graphen auch terminiert [Flo62] und [War62]. Durch den Zusammenhang kann vom metrischen Abschluss problemlos ein minimaler aufspannender Baum bestimmt werden, was zum Beispiel mit dem Algorithmus von Prim erfolgen kann, welcher ebenfalls terminiert, da es einen minimalen aufspannenden Baum gibt [Pri57]. Das anschließende Ersetzen der Kanten aus dem minimalen Baum durch entsprechende kürzeste Wege aus dem Eingabegraphen beeinflusst den Zusammenhang nicht. Es kann wieder ein minimaler aufspannender Baum für den so entstandenen Graphen bestimmt werden. Auch das abschließende Entfernen von nicht benötigten Wegen und Kanten terminiert, da die Knoten- und Kantenmengen endlich sind.

Nun wird für die Korrektheit noch gezeigt, dass der Algorithmus bei Termination einen Steinerbaum mit der Terminalmenge T liefert. Da in den ersten Schritten für den metrischen Abschluss die Terminale die Knotenmenge bilden und für diesen ein minimaler aufspannender Baum bestimmt wird, welcher als Grundlage dient, sind die Terminale schon mal in diesem Baum enthalten. Im nächsten Schritt werden die Kanten durch kürzeste Wege aus dem Eingabegraphen ersetzt. Dadurch bleiben weiterhin alle Terminale erhalten und es werden lediglich weitere Knoten und Kanten hinzugefügt. In diesem Schritt können Kreise entstehen, das ist aber kein Problem, da wieder ein minimaler aufspannender Baum bestimmt wird. Im letzten Schritt werden nur noch Knoten und Kanten entfernt. Da dabei keine Terminale entfernt werden und der Zusammenhang erhalten bleibt, handelt es sich bei der Ausgabe des Algorithmus um einen Steinerbaum mit der geforderten Terminalmenge.

4.4 Approximationsgüte

Nun wird bestimmt, wie viel schwerer ein vom Algorithmus erzeugter Steinerbaum als ein minimaler Steinerbaum für eine Eingabe (G, c, T) maximal sein kann. Dazu wird ein minimaler Steinerbaum S_{min} für die Eingabe innerhalb des Gesamtgraphen G betrachtet. Auf einem Baum lässt sich mittels Tiefensuche ein eulerscher Kreis finden, der jeden Knoten mindestens einmal und jede Kante genau zweimal durchläuft. Sei w ein solcher Kreis für S_{min}, dann sind die Kosten für w doppelt so groß, wie das Gewicht von S_{min}. Dies ist in Abbildung 4.7 beispielhaft dargestellt. Je nach Startpunkt in w ergibt sich eine Reihenfolge, in der die Knoten aus T erstmalig durchlaufen werden. In der genannten Abbildung könnte mit dem Startpunkt v_1 die Reihenfolge $v_1v_3v_6v_9$ sein.

Mit dieser Reihenfolge lässt sich ein Kreis w' im metrischen Abschluss $G_M(T)$ erzeugen (Abbildung 4.8), wobei die Kosten von w' kleiner gleich der Kosten von w sind, da im metrischen Abschluss die Distanzen der kürzesten Wege verwendet werden, während im Steinerbaum S_{min} auch längere Wege verwendet werden können.

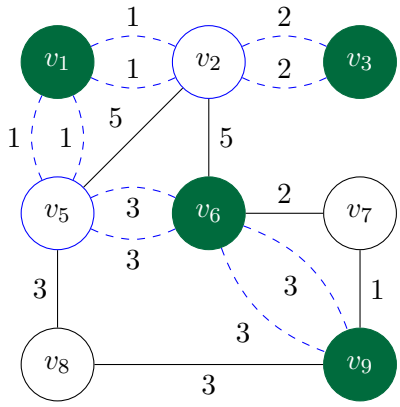


Abbildung 4.7 S_{min} mit blauen Steinerknoten und grünen Terminalen in G mit Kreis w über blaue doppelte Kanten mit $c(w) = 20$

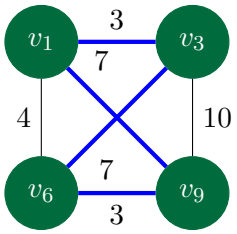


Abbildung 4.8 $w' = v_1v_3v_6v_9v_1$ in $G_M(T)$ basierend auf Kreis in S_{min} mit $c(w') = 20$

Wird nun eine Kante mit dem größten Gewicht e_{max} aus w' entfernt, so erhält man zusammen mit den Knoten T einen aufspannenden Baum B aus $G_M(T)$. Da im Kou-Markowsky-Berman-Algorithmus ein minimaler Baum B_m aus $G_M(T)$ als Grundlage für den ausgegebenen Steinerbaum S benutzt wird und in nachfolgenden Schritten die Kanten aus B_m durch kostengleiche Wege ersetzt werden, sowie gegebenenfalls überflüssige Knoten und Kanten entfernt werden, gilt: $c(S) \leq c(B_m) \leq c(B)$.

Um die Approximationsgüte zu bestimmen, wird nun das maximal mögliche Gewicht von B bestimmt. Dieses entspricht den Kosten von w' ohne die Kosten von e_{max}. Die Kosten von e_{max} müssen mindestens $\frac{1}{|T|} \cdot c(w')$ groß sein, da es genau $|T|$ Kanten im

Kreis w' gibt. Für die Kosten von B bedeutet das:

$$c(B) = c(w') - c(e_{max}) \leq c(w') - \frac{1}{|T|} \cdot c(w') = c(w') - c(w') \cdot \frac{1}{|T|} = c(w') \cdot (1 - \frac{1}{|T|})$$
$$\leq c(w) \cdot (1 - \frac{1}{|T|}) = 2 \cdot c(S_{min}) \cdot (1 - \frac{1}{|T|}) = 2 \cdot (1 - \frac{1}{|T|}) \cdot c(S_{min})$$

Damit ergibt sich für den vom Algorithmus konstruierten Steinerbaum S folgender Zusammenhang zu S_{min}:

$$c(S) \leq c(B) \leq 2(1 - \frac{1}{|T|}) \cdot c(S_{min})$$

Dies bedeutet, dass der vom Algorithmus erzeugte Steinerbaum S maximal ein $2(1 - \frac{1}{|T|})$ mal so großes Gewicht, wie ein minimaler Steinerbaum S_{min} für die Eingabe hat. Da dies kleiner als das doppelte Gewicht ist, wird der Kou-Markowsky-Berman-Algorithmus auch als 2-Approximationsalgorithmus bezeichnet.

4.5 Laufzeit

Um die Laufzeit zu bestimmen, werden die Aufwände für die einzelnen Schritte von eins bis fünf betrachtet. Für Schritt eins müssen die kürzesten Wege für den metrischen Abschluss bestimmt werden. Dies kann zum Beispiel mit dem Algorithmus von Floyd und Warshall in $\mathcal{O}(|V(G)|^3)$ geschehen [Flo62] [War62]. Für Schritt zwei muss ein minimaler aufspannender Baum konstruiert werden, was mit den Algorithmus von Prim in einer Laufzeit von $\mathcal{O}(|V(G)|^2)$ möglich ist [Pri57]. Schritt drei lässt sich in $\mathcal{O}(|V|)$ durchführen, da lediglich die im ersten Schritt berechneten kürzesten Wege eingefügt werden müssen [KMB81]. Der vierte Schritt ist äquivalent zu Schritt zwei. Zuletzt müssen im fünften Schritt noch überflüssige Wege und Kanten entfernt werden. Dies kann zum Beispiel mittels einer Breitensuche in $\mathcal{O}(|V(G)|^2)$ durchgeführt werden [KV08, 534f]. Damit ist der erste Schritt der aufwendigste und der Kou-Markowsky-Berman-Algorithmus läuft in $\mathcal{O}(|V(G)|^3)$-Zeit.

Mit der Verwendung des Dijkstra-Algorithmus ([Dij59]) kann die Laufzeit verbessert werden, indem dieser mit Fibonacci-Heaps implementiert wird. Der Dijkstra-Algorithmus hat dann eine Laufzeit von $\mathcal{O}(|E| + |V| \log |V|)$. Für die Berechnung aller benötigten kürzesten Wege muss dieser im schlechtesten Fall für alle Knoten ausgeführt werden.

Dadurch würde der Kou-Markowsky-Berman-Algorithmus in $\mathcal{O}(|V||E| + |V|^2 \log |V|)$-Zeit laufen. [Tur09, S. 262]

5 Der Algorithmus von Dreyfus und Wagner

In diesem Kapitel geht es um den Dreyfus-Wagner-Algorithmus. Dieser ist, im Gegensatz zum Approximations-Algorithmus vom Kou, Markowsky und Berman, ein exakter Algorithmus und daher auch aufwendiger. Der Algorithmus hat zwar eine exponentielle Laufzeit, aber der Exponent wird durch die Größe der Terminalmenge bestimmt, somit ist der Algorithmus für kleinere Terminalmengen noch effizient. Neben der genauen Beschreibung des Algorithmus wird hier auch die Korrektheit und Laufzeit untersucht. Grundlage für dieses Kapitel bilden [DW71], [Röh08] und [KV08].

5.1 Funktionsweise

Um die Funktionsweise des Dreyfus-Wagner-Algorithmus zu verstehen, wird zunächst ein Lemma eingeführt, auf welchem der Algorithmus im Wesentlichen basiert. Mit den Gleichungen aus diesem kann eine Instanz des Steinerbaumproblems in kleinere Teilprobleme zerlegt werden.

Betrachtet man einen Steinerbaum S für eine Terminalmenge $X \cup \{v\}$, kann man, wenn v ein Blatt ist, solange von v ausgehend über Steinerknoten laufen, bis man entweder auf ein Terminalknoten oder einen Knoten mit Knotengrad ≥ 3 stößt. S setzt sich dann aus dem durchlaufenden Weg und einem Steinerbaum mit der Terminalmenge X zusammen. Dies ist in Abbildung 5.1 beispielhaft dargestellt. Ist v hingegen ein innerer Knoten, dann kann S an v in zwei Steinerbäume mit den Terminalmengen $X' \subset X$ und $X \setminus X'$ aufgeteilt werden. Dieser Fall ist in Abbildung 5.2 dargestellt.

Nach diesem Prinzip lässt sich ein Steinerbaum solange zerlegen, bis nur noch Wege als Komponenten übrig bleiben. Der Algorithmus baut, beginnend mit kürzesten Wegen, schrittweise den minimalen Steinerbaum auf. Das Problem dabei liegt darin, dass alle möglichen Zusammensetzungen durchprobiert werden müssen, damit der Steinerbaum minimal wird, wodurch sich die exponentielle Laufzeit ergibt.

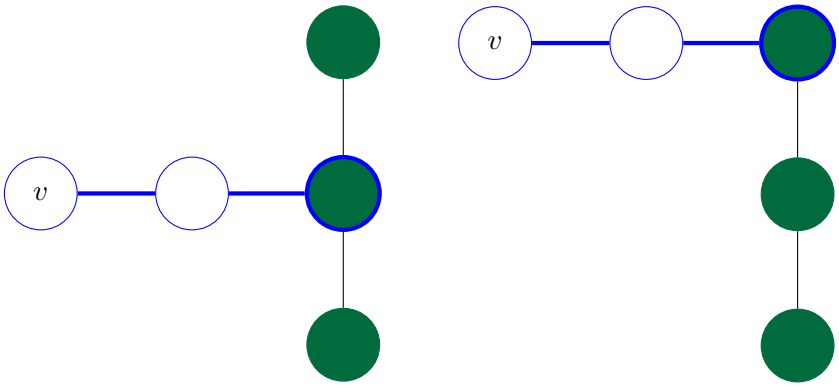


Abbildung 5.1 Zerlegung von Steinerbäumen in kleinere Steinerbäume (grün) und Wege (blau)

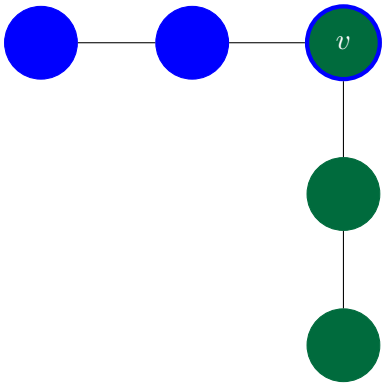


Abbildung 5.2 Aufteilung eines Steinerbaums in zwei Steinerbäume (grün und blau) am Knoten v

Lemma

Für eine Instanz des Steinerbaumproblems $G = (V, E)$, $c\colon E \to N$ und $T \subseteq V$ werden für alle $X \subseteq T$ und $v \in V \setminus X$ die folgenden Funktionen definiert:

$$minStTree(X) \coloneqq \min\{c(S) \mid S \text{ ist ein Steinerbaum für die Terminalmenge } X \text{ in } G\}$$

$$innerVertex(X, v) \coloneqq \min\{c(S) \mid S \text{ ist ein Steinerbaum für die Terminalmenge } X \cup \{v\} \text{ mit } v \text{ als inneren Knoten}\}$$

Gibt es keinen entsprechenden Steinerbaum für $innerVertex(X, v)$, dann ist das Ergebnis der Minimierung ∞.

Für alle $X \subseteq T$ mit $|X| \geq 2$ und $v \in V \setminus X$ gilt dann:

$$innerVertex(X, v) = \min_{\emptyset \neq X' \subset X} \{minStTree(X' \cup \{v\}) + minStTree((X \setminus X') \cup \{v\})\},$$
$$\text{wenn } innerVertex(X, v) \neq \infty$$

$$minStTree(X \cup \{v\}) = \min\{\min_{w \in X}\{short(v, w) + minStTree(X)\},$$
$$\min_{w \in V \setminus X}\{short(v, w) + innerVertex(X, w)\}\}$$

$X' \subset X$ bedeutet dabei, dass X' eine echte Teilmenge von X ist, also $X' \subseteq X$ und $X' \neq X$ gilt.

Beweis
Zunächst die erste Gleichung für $innerVertex$:
Für jeden minimalen Steinerbaum S mit einer Terminalmenge $T \cup \{v\}$ mit v als inneren Knoten gilt, dass v einen Knotengrad ≥ 2 hat. Daher lässt sich S am Knoten v in zwei Steinerbäume aufteilen, wobei jeder mindestens einen Terminalknoten und den Knoten v enthält. Dies ist in Abbildung 5.3 skizziert, wobei die unterschiedlich eingefärbten Knoten jeweils zusammen mit dem Knoten v die Steinerbäume darstellen. Diese Aufteilung erfolgt in der Gleichung, wobei über alle mögliche Aufteilungen minimiert wird, um die minimalen Kosten zu erhalten.

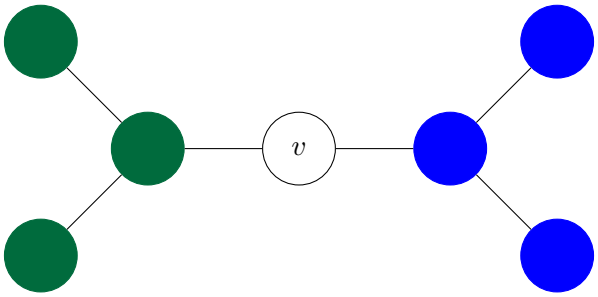


Abbildung 5.3 Aufteilung von S_{min} in zwei Steinerbäume (grün und blau) am Knoten v

Für die Gleichung $minStTree$ werden die zwei Richtungen „$\leq$“ und „$\geq$“ gezeigt:

$$minStTree(X \cup \{v\}) \leq \min\{\min_{w \in X}\{short(v,w) + minStTree(X)\},$$
$$\min_{w \in V \setminus X}\{short(v,w) + innerVertex(X,w)\}\}$$

$$minStTree(X \cup \{v\}) \geq \min\{\min_{w \in X}\{short(v,w) + minStTree(X)\},$$
$$\min_{w \in V \setminus X}\{short(v,w) + innerVertex(X,w)\}\}$$

Aus beiden Richtungen zusammen folgt die Gleichheit. Zunächst wird die erste Richtung „$\leq$“ betrachtet. Hier werden zwei Fälle unterschieden:
Im ersten Fall hat das Ergebnis der Minimierungen die Form $short(v,w)+minStTree(X)$ mit $w \in X$ (in Abbildung 5.4 dargestellt).
Dann gibt es einen Weg p von v nach w mit $c(p) = short(v,w)$ und einen minimalen Steinerbaum S_G mit der Terminalmenge X in G. Sei G' der Graph aus der Vereinigung von S_G und p, dann gilt: G' ist zusammenhängend, $G' \subseteq G$ und $X \cup \{v\} \subseteq V_{G'}$. Somit enthält G' einen Baum, der in G ein Steinerbaum $S_{G'}$ für die Terminalmenge $X \cup \{v\}$ ist. Es ergibt sich:

$$short(v,w) + minStTree(X) = c(p) + c(S_G) \geq c(G') \geq c(S_{G'}) \geq minStTree(X \cup \{v\})$$

Im zweiten Fall hat das Ergebnis der Minimierung die Form $short(v,w)+innerVertex(X,w)$ mit $w \in V \setminus X$ (in Abbildung 5.5 dargestellt).
Dann gibt es einen Weg p von v nach w mit $c(p) = short(v,w)$ und einen minimalen Steinerbaum S_G mit der Terminalmenge X und w als inneren Knoten in G. Sei G' der Graph aus der Vereinigung von S_G und p, dann gilt: G' ist zusammenhängend, $G' \subseteq G$ und $X \cup \{v\} \subseteq V_{G'}$. Somit enthält G' einen Baum, der in G ein Steinerbaum $S_{G'}$ für die Terminalmenge $X \cup \{v\}$ ist. Es ergibt sich:

$$short(v,w)+innerVertex(X,w) = c(p)+c(S_G) \geq c(G') \geq c(S_{G'}) \geq minStTree(X \cup \{v\})$$

Damit wurde die Richtung „$\leq$“ gezeigt. Für die Rückrichtung sei S_{min} ein Steinerbaum für die Terminalmenge $X \cup \{v\}$ mit $c(S_{min}) = minStTree(X \cup \{v\})$. Es folgt eine Fallunterscheidung bezüglich des Knotengrades von v in S_{min}.
Im ersten Fall hat v einen Knotengrad ≥ 2. Dann ist v ein innerer Knoten in S_{min} und

es gilt somit nach Definition von $innerVertex$:

$$c(S_{min}) = innerVertex(X, v)$$

Wir erhalten für $w = v$:

$$minStTree(X \cup \{v\}) = c(S_{min}) = short(v, v) + innerVertex(X, v)$$
$$\geq \min \left\{ \min_{w \in X} \{short(v, w) + minStTree(X)\}, \min_{w \in V \setminus X} \{short(v, w) + innerVertex(X, w)\} \right\}$$

Im zweiten Fall hat v einen Knotengrad von eins und ist damit ein Blatt in S_{min}. Betrachtet man den Knoten $w \in V(S_{min})$ der in S_{min}

- den kürzesten Weg zum Knoten v hat und dabei
- einen Knotengrad ≥ 3 hat oder in X enthalten ist,

können auch hier zwei Fälle unterschieden werden: $w \in X$ und $w \notin X$.
Ist $w \in X$, so lässt sich S_{min} in einen Steinerbaum S' für die Terminalmenge X und dem Weg p von v nach w aufteilen. Daher gilt:

$$minStTree(X \cup \{v\}) = c(S_{min}) = c(p) + c(S') \geq short(v, w) + minStTree(X)$$
$$\geq \min \left\{ \min_{w \in X} \{short(v, w) + minStTree(X)\}, \min_{w \in V \setminus X} \{short(v, w) + innerVertex(X, w)\} \right\}$$

Dieser Fall ist in Abbildung 5.4 skizziert, wobei sich zwischen v und w noch weitere Knoten, die nicht in X enthalten sind und in S_{min} einen Knotengrad von 2 haben, befinden können. w kann dabei sowohl ein Blattknoten (linker Graph), als auch ein innerer Knoten (rechter Graph) von S' sein.

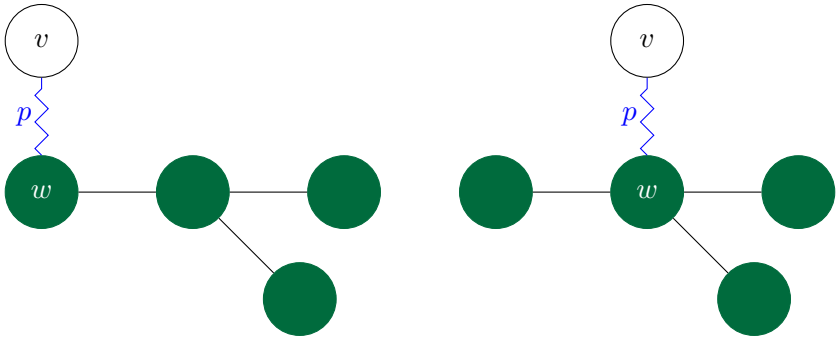


Abbildung 5.4 Anbindung von v über p an den Knoten $w \in X$

Ist $w \notin X$, hat w einen Knotengrad ≥ 3. Damit lässt sich S_{min} in einen Steinerbaum S' für die Terminalmenge $X \cup \{w\}$ mit w als inneren Knoten und einem Weg p von v nach w aufteilen und es gilt:

$$minStTree(X \cup \{v\}) = c(S_{min}) = c(p) + c(S') \geq short(v,w) + innerVertex(X,w)$$
$$\geq \min\left\{\min_{w \in X}\{short(v,w) + minStTree(X)\}, \min_{w \in V \setminus X}\{short(v,w) + innerVertex(X,w)\}\right\}$$

In Abbildung 5.5 ist dieser Fall skizziert. w ist dabei ein Steinerknoten und zwischen v und w können sich auch hier weitere Knoten, die nicht in X enthalten sind und einen Knotengrad von 2 haben, befinden.

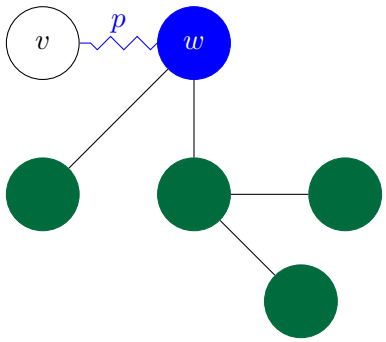


Abbildung 5.5 Anbindung von v über p an den inneren Knoten w

Damit wurde die Richtung „$\geq$“ auch gezeigt, wodurch die Gleichheit gezeigt wurde. □

Eine Instanz des Steinerbaumproblems kann mittels der Gleichungen aus dem Lemma solange in immer kleinere Teilprobleme zerlegt werden, bis nur noch Probleme der Form $minStTree(X)$ mit $|X| = 2$ und kürzeste Wege gelöst werden müssen. Da das Steinerbaumproblem für eine zweielementige Terminalmenge äquivalent zum Kürzeste-Wege-Problem ist, lassen sich diese mit entsprechenden Algorithmen einfach lösen ([Dij59], [Flo62] und [War62]). Diese Tatsache wird im Dreyfus-Wagner-Algorithmus genutzt. Hier wird das Lemma umgekehrt angewendet. Das heißt es wird zunächst das Kürzeste-Wege-Problem für alle Knotenpaare des Eingabegraphen gelöst, was den minimalen Steinerbäumen für zweielementige Terminalmengen entspricht. Anschließend werden rekursiv unter Anwendung der Gleichungen aus dem Lemma die Kosten der minimalen Steinerbäume für immer größere Terminalmengen berechnet, bis schließlich das eigentlich zu lösende Steinerbaumproblem für die eingegebene Terminalmenge gelöst wurde. Dieses Verfahren nennt sich dynamische Optimierung mit Vorwärtsrekursion [Dom+15].

Im Folgenden ist der Algorithmus in Pseudocode aufgeschrieben. Er berechnet für eine Eingabe $G = (V, E)$ zusammenhängend, $c\colon E \to N$ und $T \subseteq V$ mit $|T| \geq 2$ die Kosten eines minimalen Steinerbaums:

Der Algorithmus von Dreyfus und Wagner

//Initialisierung:

forall $v, v' \in V$ **do**

  Setze $minStTree(\{v, v'\}) = shortestPath(v, v')$

//Vorwärtsrekursion

for $i = 2$ **to** $i = |T| - 1$ **do**

  forall $X \subseteq T$ *mit* $|X| = i$, $v \in V \setminus X$ **do**

    $innerVertex(X, v) =$

      $\min_{\emptyset \neq X' \subset X} \{minStTree(X' \cup \{v\}) + minStTree((X \setminus X') \cup \{v\})\}$

  forall $X \subseteq T$ *mit* $|X| = i$, $v \in V \setminus X$ **do**

    $minStTree(X \cup \{v\}) =$

$$\min \left\{ \begin{array}{l} \min_{w \in X} \{shortestPath(v, w) + minStTree(X)\}, \\ \\ \min_{w \in V \setminus X} \{shortestPath(v, w) + innerVertex(X, w)\} \end{array} \right\}$$

Output: $minStTree(T)$

$shortestPath(v, v')$ ist hier ein Platzhalter für den Aufruf eines Algorithmus zur Berechnung der Kosten eines kürzesten Weges von v nach v' in einem Graphen. Dies kann zum Beispiel der Dijkstra-Algorithmus([Dij59]) oder der Floyd-Warshall-Algorithmus([Flo62] und [War62]) sein. Für $innerVertex(X, v)$ werden im Algorithmus auch dann Werte errechnet, wenn es keinen Graphen mit v als inneren Knoten gibt. Dies ist aber nicht weiter problematisch, da in diesen Fällen immer mindestens eine Kante doppelt eingerechnet wird, wie in Abbildung 5.6 skizziert. Dadurch sind diese Ergebnisse niemals kleiner als die Kosten eines minimalen Steinerbaums für die Terminalmenge $X \cup \{v\}$.

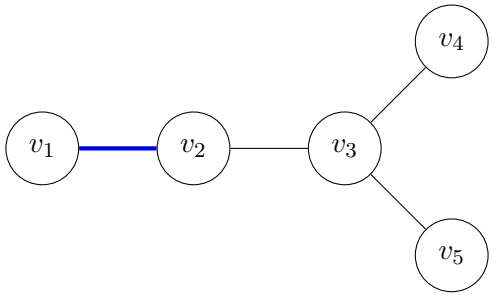


Abbildung 5.6 Bei Addieren der Kosten von zwei Steinerbäumen, die beide v_1 enthalten, wird mindestens die blaue Kante doppelt eingerechnet

5.2 Der Algorithmus am Beispiel

Der Algorithmus wird nun an einem Beispiel vorgeführt. Der in Abbildung 5.7 abgebildete Graph G mit Kantengewichten dient als Ausgangsgraph. Die Terminalmenge ist $\{v_2, v_3, v_4, v_6\}$.

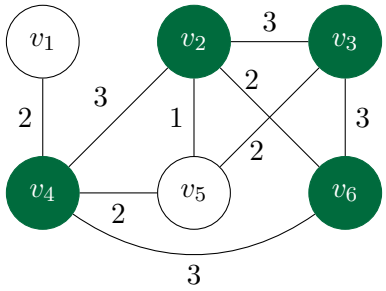


Abbildung 5.7 Ausgangsgraph G mit Terminalmenge in grün

Zunächst werden die Längen der kürzesten Wege zwischen je zwei Knoten aus $V(G)$ berechnet. Die Ergebnisse sind in Tabelle 5.1 festgehalten.

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	5	6	2	4	5
v_2	5	0	3	3	1	2
v_3	6	3	0	4	2	3
v_4	2	3	4	0	2	3
v_5	4	1	2	2	0	3
v_6	5	2	3	3	3	0

Tabelle 5.1 Distanzen kürzester Wege zwischen je zwei Knoten aus $V(G)$

Nun wird die Funktion $minStTree$ für alle zweielementigen Teilmengen von $V(G)$ mit den berechneten Distanzen initialisiert. Anschließend startet die Schleife mit den rekursiven Funktionen. Im ersten Durchgang ist $i = 2$. Zunächst werden daher die zweielementigen Teilmengen von T betrachtet. Sei $\{v_2, v_3\}$ die erste Teilmenge und v_1 der erste Knoten in der Schleife. Wird dies jetzt in $innerVertex$ eingesetzt, ergibt sich Folgendes:

$$\begin{aligned} innerVertex(\{v_2, v_3\}, v_1) &= \min\{minStTree(\{v_2\} \cup \{v_1\}) + minStTree(\{v_3\} \cup \{v_1\})\} \\ &= 5 + 6 = 11 \end{aligned}$$

Da es nur eine Aufteilung einer zweielementigen Menge in zwei nicht leere echte Teilmengen gibt, braucht hier kein Minimum ausgewählt werden. Die Ergebnisse für $minStTree$ ergeben sich durch die Initialisierung. v_1 ist in diesem Fall ein Blattknoten in G, daher gibt es keinen Steinerbaum mit v_1 als inneren Knoten. Wie bereits beschrieben, führt der Algorithmus trotzdem eine Berechnung durch, dessen Ergebnis aber nicht die Kosten eines minimalen Steinerbaums sein können, da in diesem Fall mindestens die Kante $\{v_1, v_4\}$ doppelt eingerechnet wird. Die Berechnung wird mit allen weiteren $v \in V(G) \setminus \{v_2, v_3\}$ durchgeführt. Die Ergebnisse sind in folgender Tabelle aufgelistet:

$\cup$	$\{v_2, v_3\}$
v_1	11
v_4	7
v_5	3
v_6	5

Tabelle 5.2 $innerVertex$ für $\{v_2, v_3\}$ für alle $v \in V(G) \setminus \{v_2, v_3\}$

Das Ganze wird für alle weiteren zweielementigen Teilmengen von T wiederholt. Tabelle 5.3 zeigt die erweiterte Tabelle. Für den Fall, dass ein Knoten bereits in einer Teilmenge enthalten ist wird die Funktion vom Algorithmus nicht aufgerufen, daher steht in diesen Zellen als Wert -.

$\cup$	$\{v_2, v_3\}$	$\{v_2, v_4\}$	$\{v_2, v_6\}$	$\{v_3, v_4\}$	$\{v_3, v_6\}$	$\{v_4, v_6\}$
v_1	11	7	10	8	11	7
v_2	-	-	-	6	5	5
v_3	-	7	6	-	-	7
v_4	7	-	6	-	7	-
v_5	3	3	4	4	5	5
v_6	5	5	-	6	-	-

Tabelle 5.3 $innerVertex$ für alle zweielementigen Teilmengen von T und $v \in V(G)$

Damit ist die Schleife zur Berechnung von $innerVertex$ für die erste Iteration abgeschlossen. Als nächstes wird für alle zweielementigen Teilmengen von T, kombiniert mit einem weiteren Knoten $v \in V(G)$, $minStTree$ berechnet. Sei die erste Teilmenge wieder $\{v_2, v_3\}$ und der erste Knoten v_1, lässt sich $minStTree(\{v_2, v_3\} \cup \{v_1\})$ mit den in den vorherigen Schritten berechneten Ergebnissen für $shortestPath$ und $innerVertex$, sowie der initialisierten Funktion $minStTree$ bestimmen:

$$
\begin{aligned}
minStTree(\{v_2, v_3\} \cup \{v_1\}) = \min \Bigg\{ & \min \begin{Bmatrix} shortestPath(v_1, v_2) + minStTree(\{v_2, v_3\}), \\ shortestPath(v_1, v_3) + minStTree(\{v_2, v_3\}) \end{Bmatrix}, \\
& \min \begin{Bmatrix} shortestPath(v_1, v_1) + innerVertex(\{v_2, v_3\}, \{v_1\}), \\ shortestPath(v_1, v_4) + innerVertex(\{v_2, v_3\}, \{v_4\}), \\ shortestPath(v_1, v_5) + innerVertex(\{v_2, v_3\}, \{v_5\}), \\ shortestPath(v_1, v_6) + innerVertex(\{v_2, v_3\}, \{v_6\}) \end{Bmatrix} \Bigg\} \\
= \min & \{\min\{8, 9\}, \min\{11, 9, 7, 10\}\} \\
= 7 &
\end{aligned}
$$

Äquivalent berechnen sich die Werte für alle weiteren Kombinationen:

$\cup$	$\{v_2, v_3\}$	$\{v_2, v_4\}$	$\{v_2, v_6\}$	$\{v_3, v_4\}$	$\{v_3, v_6\}$	$\{v_4, v_6\}$
v_1	7	5	7	6	8	5
v_2	-	-	-	5	5	5
v_3	-	5	5	-	-	6
v_4	5	-	5	-	6	-
v_5	3	3	3	4	5	5
v_6	5	5	-	6	-	-

Tabelle 5.4 *minStTree* für alle zweielementigen Teilmengen von T und $v \in V(G)$

Die berechneten Werte entsprechen den Größen der minimalen Steinerbäume für die jeweiligen dreielementigen Terminalmengen in G. Der erste Durchgang der Schleife ist damit abgeschlossen. Nun ist $i = 3$ und die Berechnungen für alle dreielementigen Teilmengen von T mit einem zusätzlichen Knoten aus $V(G)$ werden durchgeführt. Für $innerVertex$ ergibt sich dabei Folgendes:

$\cup$	$\{v_2, v_3, v_4\}$	$\{v_2, v_3, v_6\}$	$\{v_2, v_4, v_6\}$	$\{v_3, v_4, v_6\}$
v_1	9	12	9	10
v_2	-	-	-	7
v_3	-	-	8	-
v_4	-	8	-	-
v_5	5	5	5	7
v_6	8	-	-	-

Tabelle 5.5 *innerVertex* für alle dreielementigen Teilmengen von T und $v \in V(G)$

Mithilfe dieser Werte lassen sich nun wieder die Berechnungen für $minStTree$ durchführen. Diesmal für dreielementige Teilmengen von T mit einem weiteren Knoten $v \in V(G)$:

$\cup$	$\{v_2, v_3, v_4\}$	$\{v_2, v_3, v_6\}$	$\{v_2, v_4, v_6\}$	$\{v_3, v_4, v_6\}$
v_1	7	9	7	8
v_2	-	-	-	7
v_3	-	-	7	-
v_4	-	7	-	-
v_5	5	5	5	7
v_6	7	-	-	-

Tabelle 5.6 $minStTree$ für alle dreielementigen Teilmengen von T und $v \in V(G)$

Damit terminiert der Algorithmus. Die Ausgabe ist $minStTree(T)$, welche aus der Tabelle aus den gelb markierten Zellen ablesbar ist. Damit hat ein minimaler Steinerbaum für die Terminalmenge T in G Kosten in Höhe von sieben. Ein entsprechender minimaler Steinerbaum ist leicht in G zu erkennen:

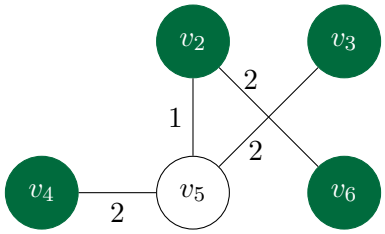


Abbildung 5.8 Minimaler Steinerbaum für die Terminalmenge T in G mit v_5 als Steinerknoten

5.3 Korrektheit

Zunächst wird gezeigt, dass der Algorithmus für alle Eingaben mit $|T| \geq 2$ terminiert. Anschließend wird gezeigt, dass bei Termination die Kosten eines minimalen Steinerbaums für die Eingabe ausgegeben werden.
Für die Initialisierung werden die Kosten der kürzesten Wege für alle Knotenpaare im Ausgangsgraphen G berechnet. Da G eine endliche Anzahl an Knoten hat und zusammenhängend ist, können die Kosten aller kürzesten Wege mittels Algorithmen, wie den Dijkstra- oder den Floyd-Warshall-Algorithmus berechnet werden, welche ebenfalls terminieren ([Dij59],[Flo62] und [War62]). Durch die Endlichkeit des Eingabegraphen sind die Anzahl der Teilmengen von T, sowie die Möglichkeiten für $v \in V \setminus X$ begrenzt und damit auch die Anzahl der Schleifendurchgänge und den enthaltenen Berechnungen. Für den ersten Schleifendurchgang wurde $minStTree$ für alle Knotenpaare mit den Kosten der kürzesten Wege initialisiert. Dadurch können auch alle Aufrufe dieser Funktion im ersten Durchgang aufgelöst werden. Für die nachfolgenden Schleifendurchgänge werden dann die Ergebnisse aus dem vorherigen Durchgängen benutzt. Da somit alle Berechnungen durchgeführt werden können, terminiert der Algorithmus, sobald der letzte Schleifendurchgang mit $i = |T| - 1$ beendet wurde.
Die Termination wurde damit gezeigt. Für die Korrektheit wird der folgendes Satz bewiesen.

Satz:
Der Dreyfus-Wagner-Algorithmus berechnet die Kosten von minimalen Steinerbäumen für alle Terminalmengen T mit $|T| \geq 2$.

Beweis:
Dies lässt sich mittels Induktion zeigen. Zunächst werden Instanzen des Steinerbaum-

problems mit zweielementigen Terminalmengen betrachtet. Geht man davon aus, dass für die Initialisierung ein Algorithmus verwendet wird, der die Kosten der kürzesten Wege korrekt berechnet, so wurden auch die Kosten aller minimalen Steinerbäume mit zweielementigen Terminalmengen korrekt bestimmt und damit auch für die eingegebene Instanz des Steinerbaumproblems. Die Kosten für Steinerbäume mit Terminalmengen X mit $|X| = 2 = n$ werden also korrekt berechnet. Dies ist der Induktionsanfang.
Nun werden Instanzen des Steinerbaumproblems mit Terminalmengen X der Größe $n+1$ betrachtet. Die Induktionsvoraussetzung ist die korrekte Berechnung der Kosten der minimalen Steinerbäume für Terminalmengen $X' \subset X$ mit $2 \leq |X'| \leq n$. Der Algorithmus berechnet mit den Gleichungen aus dem im Abschnitt 5.1 eingeführten Lemma und den Ergebnissen der Induktionsvoraussetzung die Kosten der minimalen Steinerbäume für alle Terminalmengen $X' \cup \{v\}$ mit $v \in V \setminus X'$ und $|X' \cup \{v\}| = n+1$ und damit die Kosten für die Terminalmengen X der Größe $n+1$. Dass diese Berechnungen korrekt sind, folgt aus der Korrektheit der Gleichungen aus dem Lemma, die bereits gezeigt wurde. Damit ist der Induktionsschritt abgeschlossen und die Korrektheit gezeigt. □

5.4 Laufzeit

Die Gesamtlaufzeit für den Algorithmus ergibt sich aus der Laufzeit für die Berechnungen der kürzesten Wege in der Initialisierung und den Aufwänden für die Berechnungen von $innerVertex$ und $minStTree$. Sei $n = |V(G)|$ und $t = |T|$. Für die Berechnung der kürzesten Wege für alle Knotenpaare kann der Floyd-Warshall-Algorithmus verwendet werden. Dieser hat eine Laufzeit von $\mathcal{O}(n^3)$. [Flo62] und [War62]

Als nächstes wird der Aufwand für die Berechnungen von $innerVertex$ abgeschätzt. Die oberste Schleife in der Rekursion iteriert von $i = 2$ bis $t-1$ und ruft in jedem Durchgang die Schleife zur Berechnung von $innerVertex$ auf, welche alle i-elementigen Teilmengen $X \subset T$ kombiniert mit einem weiteren Knoten durchgeht. Von T gibt es $\binom{t}{i}$ i-elementige Teilmengen. Damit ergibt sich

$$\sum_{i=2}^{t-1} \binom{t}{i} \cdot n$$

Für die Berechnung von $innerVertex$ werden alle Möglichkeiten durchgegangen, X in zwei Mengen aufzuteilen. Von X gibt es $2^i - 2$ echte Teilmengen und damit Möglichkeiten

für die Aufteilungen. Der Gesamtaufwand ist damit

$$\sum_{i=2}^{t-1} \binom{t}{i} \cdot n \cdot (2^i - 2) \leq \sum_{i=0}^{t} \binom{t}{i} \cdot 2^i \cdot n$$

Mithilfe des Binomischen Lehrsatzes lässt sich dieser Ausdruck noch weiter vereinfachen. Der Lehrsatz sieht wie folgt aus:

$$\sum_{i=0}^{n} \binom{n}{i} x^{n-i} y^i = (x+y)^n$$

Um den Lehrsatz anzuwenden wird der Ausdruck noch um den Faktor 1^{t-i} erweitert. Dies ist möglich, da dessen Ergebnis immer eins ist. Insgesamt ergibt sich damit folgender Aufwand für $innerVertex$:

$$\sum_{i=0}^{t} \binom{t}{i} \cdot 1^{t-i} \cdot 2^i \cdot n = (1+2)^t \cdot n = 3^t \cdot n$$

Für $minStTree$ wird der Aufwand auf gleiche Weise abgeschätzt. Es beginnt wieder mit der äußersten Rekursionsschleife von $i = 2$ bis $t - 1$, welche in jedem Durchgang die Schleife zur Berechnung von $minStTree$ aufruft. Hier werden wieder in jedem Durchgang alle i-elementigen Teilmengen kombiniert mit einem weiteren Knoten durchgegangen. In der Berechnung für $minStTree$ werden nochmals alle Möglichkeiten für den Knoten w durchgegangen. Insgesamt ergibt sich damit

$$\sum_{i=2}^{t-1} \binom{t}{i} \cdot n \cdot n \leq \sum_{i=0}^{t} \binom{t}{i} \cdot n^2$$

Auch hier kann der Ausdruck wieder einfach erweitert werden, damit der Lehrsatz angewendet werden kann:

$$\sum_{i=0}^{t} \binom{t}{i} \cdot n^2 = \sum_{i=0}^{t} \binom{t}{i} \cdot 1^{t-i} \cdot 1^i \cdot n^2 = (1+1)^t \cdot n^2 = 2^t \cdot n^2$$

Als Gesamtlaufzeit für den Dreyfus-Wagner-Algorithmus ergibt sich damit:

$$\mathcal{O}(3^t n + 2^t n^2 + n^3)$$

Die Laufzeit hängt damit im wesentlichen von der Anzahl der Terminale ab. Für eine

konstante Anzahl an Terminalen läuft der Algorithmus in polynomieller Zeit. Ist $3^t \leq n^2$ und $2^t \leq n$ dann ist der Aufwand kubisch. Umgestellt bedeutet das für die Anzahl der Terminale: $t \leq \log_3(n^2)$ und $t \leq \log_2(n)$. In Abbildung 5.9 ist dieser Zusammenhang mittels der Funktionen $t_1 = \log_3(n^2)$ und $t_2 = \log_2(n)$ graphisch aufgetragen. Auf der X-Achse ist die Anzahl der Gesamtknoten aufgetragen und auf der Y-Achse die Anzahl der Terminalknoten. Dadurch lässt sich in Abhängigkeit der Gesamtknoten die maximale Anzahl an Terminalknoten ablesen, bei der der Dreyfus-Wagner-Algorithmus einen kubischen Aufwand hat. Es wird deutlich, dass t_2 der kritische Wert ist, der die maximale Anzahl an Terminalknoten bestimmt.

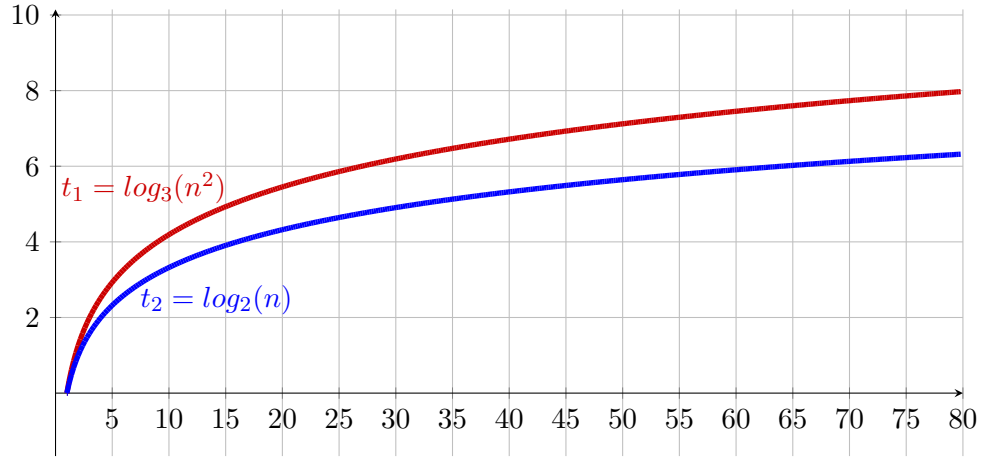


Abbildung 5.9 Darstellung der Funktionen als Graphen mit Anzahl der Knoten auf der X-Achse und Anzahl der Terminalknoten auf der Y-Achse, erstellt mit GeoGebra [Geo21]

Wird statt des Floyd-Warshall-Algorithmus der Dijkstra-Algorithmus ([Dij59]) verwendet, beeinflusst das die Laufzeit kaum, da die Laufzeit der Rekursionsschleife meistens überwiegt. Wird der Dijkstra-Algorithmus mit Fibonacci-Heaps implementiert, dann läuft dieser in $\mathcal{O}(m + n \log n)$, wobei $m = |E|$ ist [Tur09, S. 262]. Der Algorithmus muss aber für die Berechnung der kürzesten Wege aller Knotenpaare n-mal ausgeführt werden. Damit ergibt sich als Gesamtlaufzeit für den Dreyfus-Wagner-Algorithmus:

$$\mathcal{O}(3^t n + 2^t n^2 + mn + n^2 \log n)$$

5.5 Erweiterung des Algorithmus für die Ausgabe eines minimalen Steinerbaums

Der Algorithmus von Dreyfus und Wagner bestimmt die Kosten eines minimalen Steinerbaums für eine Eingabe. Daher folgen nun ein paar Überlegungen, wie der Algorithmus sich leicht erweitern lässt, damit auch ein minimaler Steinerbaum berechnet werden kann. Dazu könnte beispielsweise bei jedem Berechnungsschritt die Knoten- und Kantenmenge gespeichert werden, mit welcher das Ergebnis zustande gekommen ist, angefangen damit, bei der Initialisierung nicht nur die Kosten der kürzesten Wege, sondern auch die Wege an sich zu bestimmen. Damit hätte man am Ende auch die Knoten und Kanten für einen minimalen Steinerbaum. Allerdings würde dieses Vorgehen viel Speicher verbrauchen, da für jede Berechnung von $innerVertex$ und $minStTree$ ein Baum gespeichert werden würde, was abhängig von der Terminalmenge exponentiell viele Bäume wären.

Eine bessere Methode wäre es, für jedes Zwischenergebnis immer nur die hinzugefügten Knoten und Kanten zu speichern und auf vorherige Ergebnisse zu referenzieren, statt den gesamten Baum zu speichern. So kann der minimale Steinerbaum, nachdem die Schleifen durchlaufen sind, aus den einzelnen Zwischenergebnissen schrittweise aufgebaut werden. In jeder Berechnung von minStTree müsste nur noch eine Referenz und ein Weg gespeichert werden und bei der Berechnung von $innerVertex$ zwei Referenzen.

Noch weniger Speicher würde man benötigen, wenn man entweder nur die hinzugefügten Kanten oder nur die hinzugefügten Knoten speichert, denn die Kanten geben auch die Knotenmenge für den Baum vor. Hat man nur die Knoten gespeichert, so kann man den minimalen aufspannenden Baum für den Teilgraphen, der genau diese Knotenmenge und alle Kanten zwischen diesen aus dem Eingabegraphen enthält, bestimmen. Dieser Baum entspricht einem gesuchten minimalen Steinerbaum. Dieses Vorgehen benötigt zwar am wenigsten Speicher, dafür ist das Erstellen des Ausgabebaums aufwendiger.

6 Zusammenfassung und Ausblick

Da diese Arbeit das Steinerbaumproblem und zwei ausgewählte Algorithmen möglichst einfach verständlich näherbringen soll, gab es zunächst eine Einführung in die benötigten Grundlagen der Graphentheorie. Darauf aufbauend wurden Steinerbäume definiert und an Beispielen vorgestellt, sowie die Problemstellung dargelegt. Zudem wurde die NP-Vollständigkeit des Steinerbaumproblems mittels einer Reduktion von Vertex-Cover auf das Problem bewiesen. Als erster Algorithmus wurde anschließend der Approximations-Algorithmus von Kou, Markowsky und Berman behandelt. Die Funktionsweise wurde mit Beispielen erläutert, die Korrektheit gezeigt und die Laufzeit $\mathcal{O}(|V(G)|^3)$ ermittelt. Für diesen Algorithmus wurde zusätzlich die Approximationsgüte von $2(1 - \frac{1}{|T|})$ bestimmt und bewiesen.

Als weiterer Algorithmus war der Dreyfus-Wagner-Algorithmus Thema dieser Arbeit. Auch hier wurde die Erklärung der Funktionsweise mit einem Beispiel unterstützt und die Korrektheit gezeigt, sowie die Laufzeit $\mathcal{O}(3^{|T|}|V|+2^{|T|}|V|^2+|V|^3)$ bestimmt. Mit dem Dreyfus-Wagner-Algorithmus wurde damit neben dem Approximations-Algorithmus noch ein exakter, dafür deutlich aufwendiger Lösungsalgorithmus für das Steinerbaumproblem behandelt. Die ausführlichen Erklärungen und Beispiele gestalten dabei die Funktionsweise der Algorithmen nachvollziehbar. Auch die Beweise wurden durch Skizzen unterstützt und detaillierter beschrieben, als es in der genutzten Literatur der Fall ist.

Diese Arbeit hat sich mit zwei der bekanntesten Algorithmen beschäftigt. Es gibt aber noch einige weitere Algorithmen und auch verbesserte Varianten der vorgestellten Algorithmen, welche noch näher betrachtet werden könnten. So hat zum Beispiel Mehlhorn die Laufzeit vom Kou-Markowsky-Berman-Algorithmus auf $\mathcal{O}(|V| \log |V| + |E|)$ verbessern können, indem statt eines metrischen Abschlusses ein Graph berechnet wird, dessen minimale aufspannende Bäume auch minimale aufspannende Bäume des metrischen Abschlusses sind. [Meh88] [KV08, S. 535]

Basierend auf der Idee vom Dreyfus-Wagner-Algorithmus wurde von Fuchs, Kerner und Wang ein Algorithmus mit einer Laufzeit von $\mathcal{O}(2,684^{|T|})$ entwickelt, indem sie die Eigenschaft nutzen, dass minimale Steinerbäume in drei minimale Teilbäume, die eben-

falls minimale Steinerbäume für ein Teil der Terminalmenge sind, aufgeteilt werden können [FKW07]. Mölle, Richter und Rossmanith haben einen weiteren exakten Algorithmus, der ebenfalls auf dynamischer Optimierung basiert, mit einer Laufzeit von $\mathcal{O}((2+\delta)^{|T|} \cdot poly(n))$ für ein beliebiges aber festes $\delta > 0$ entwickelt [MRR06].

Eine Reihe weiterer Algorithmen stammen von Zelikovsky. So hat dieser den 11/6-Approximations-Algorithmus entwickelt. Der Name leitet sich von der Approximationsgüte, die $\frac{11}{6}$ des optimalen Steinerbaums beträgt, ab. Der Algorithmus verwendet eine in der Laufzeit verbesserte Variante des Kou-Markowsky-Berman-Algorithmus, optimiert aber vorher die Eingabe für diesen, indem die Terminalmenge um weitere Knoten erweitert wird und damit bereits Steinerknoten vorgegeben werden. Dadurch wird die verbesserte Approximationsgüte erreicht bei einer Laufzeit von $\mathcal{O}(|V||E| + |T|^4)$. [Zel93]

Ebenfalls von Zelikovsky stammt ein Greedy-Approximations-Algorithmus, welcher auch als Relative-Greedy-Algorithmus bezeichnet wird. In diesem wird eine Bewertungsfunktion benutzt, mit der kleinere Steinerbäume mit einer vorher festgelegten maximalen Anzahl an Terminalen bestimmt werden. Aus der Kombination der bestimmten Bäume, welche als Komponenten bezeichnet werden, wird der Steinerbaum für die gesamte Terminalmenge erzeugt. Der Algorithmus nutzt dabei die Tatsache, dass für Terminalmengen mit konstanter Größe die Steinerbäume in polynomieller Zeit berechnet werden können. Zelikovsky gab eine Approximationsgüte kleiner als $1,694$ an. Die genaue Approximationsgüte ist aber unbekannt. Hougardy und Kirchner haben aber als untere Grenze den Wert $1,385$ bestimmt. [Zel96] [HK06]

Zusammen mit Robins hat Zelikovsky den Loss-Contracting-Algorithmus entwickelt. Dieser funktioniert ähnlich wie der Relative-Greedy-Algorithmus, hat aber eine optimierte Wahl der Komponenten. Robins und Zelikovsky haben die Approximationsgüte mit kleiner als $1,55$ bestimmt. Später konnte als untere Grenze für die Approximationsgüte der Wert $1,2$ bestimmt werden. [RZ00] [Grö+01]

Neben der Behandlung weiterer Algorithmen wäre sicherlich auch die praktische Umsetzung und Visualisierung einiger Algorithmen interessant. So könnten die Algorithmen nicht nur interaktiv ausprobiert werden, sondern zum Beispiel auch die Performance gemessen werden. Auch interessant wäre eine Betrachtung des Steinerbaumproblems für spezielle Graphen, wie gerichtete Graphen, bipartite Graphen, metrische Graphen oder unendliche Graphen [KV08] [Die06].

Literaturverzeichnis

[Cyc21] CyclOSM. *CyclOSM: OpenStreetMap-based bicycle map*. 2021. URL: *https://www.cyclosm.org/* (abgerufen am 30. 08. 2021).

[Die06] Reinhard Diestel. *Graphentheorie*. Springer, 2006.

[Dij59] E. W. Dijkstra. »A note on two problems in connexion with graphs«. In: 1 (1959), S. 269–271.

[Dom+15] Wolfgang Domschke u. a. »Dynamische Optimierung«. In: *Einführung in Operations Research*. Springer Berlin Heidelberg, 2015, S. 165–182.

[DW71] S. E. Dreyfus und R. A. Wagner. »The steiner problem in graphs«. In: *Networks* 1.3 (1971), S. 195–207.

[Flo62] Robert W. Floyd. »Algorithm 97: Shortest Path«. In: *Commun. ACM* 5.6 (Juni 1962), S. 345.

[FKW07] Bernhard Fuchs, Walter Kern und Xinhui Wang. »Speeding up the Dreyfus-Wagner Algorithm for minimum Steiner trees«. In: *Mathematical Methods of Operations Research volume 66 (2007)* (2007), S. 117–125.

[Geo21] GeoGebra. *GeoGebra Classic*. 2021. URL: *https://www.geogebra.org/classic* (abgerufen am 31. 08. 2021).

[Grö+01] Clemens Gröpl u. a. »Lower Bounds for Algorithms for the Steiner Tree Problem«. In: *Graph-Theoretic Concepts in Computer Science*. Hrsg. von Andreas Brandstädt und Van Bang Le. Springer Berlin Heidelberg, 2001, S. 217–228.

[HK06] Stefan Hougardy und Stefan Kirchner. »Lower Bounds for the Relative Greedy Algorithm for Approximating Steiner Trees«. In: *Networks 47:2 (2006)* (2006), S. 111–115.

[KV08] Bernhard Korte und Jens Vygen. *Kombinatorische Optimierung: Theorie und Algorithmen*. Springer, 2008.

[KMB81] L. Kou, G. Markowsky und L. Berman. »A Fast Algorithm for Steiner Trees«. In: *Acta Informatica 15 (1981)* (1981), S. 141–145.

[Kru56] Joseph B. Kruskal. »On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem«. In: *Proceedings of the American Mathematical Society* 7.1 (1956), S. 48–50.

[Kus20] Sabine Kuske. *Algorithmen auf Graphen*. Vorlesungsskript, Universität Bremen. 2020.

[Lie16] Jens Lienig. *Layoutsynthese elektronischer Schaltungen*. Springer Vieweg, 2006/2016.

[Meh88] Kurt Mehlhorn. »A faster approximation algorithm for the Steiner problem in graphs«. In: *Information Processing Letters* 27.3 (1988), S. 125–128.

[MRR06] Daniel Mölle, Stefan Richter und Peter Rossmanith. »A Faster Algorithm for the Steiner Tree Problem«. In: *STACS 2006*. Hrsg. von Bruno Durand und Wolfgang Thomas. Springer Berlin Heidelberg, 2006, S. 561–570.

[Mün15] Technische Universität München. *Das Steinerbaumproblemn*. 2015. URL: *https://www-m9.ma.tum.de/games/steiner-game/index_de.html* (abgerufen am 19. 05. 2021).

[Pri57] R. C. Prim. »Shortest connection networks and some generalizations«. In: *The Bell System Technical Journal* 36.6 (1957), S. 1389–1401.

[RZ00] Gabriel Robins und Alexander Zelikovsky. »Improved Steiner Tree Approximation in Graphs«. In: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '00. Society for Industrial und Applied Mathematics, 2000, S. 770–779.

[Röh08] Linda Röhrkohl. *Das Steinerbaumproblem - Ein klassisches Problem der Graphentheorie*. aus „Seminar und Proseminar - WS 2007/2008 - Perlen der Theoretischen Informatik“ von Friedhelm Meyer auf der Heide. Universität Paderborn. 2008.

[Tur09] Volker Turau. *Algorithmische Graphentheorie*. Oldenbourg Verlag München, 2009.

[War62] Stephen Warshall. »A Theorem on Boolean Matrices«. In: *J. ACM* 9.1 (Jan. 1962), S. 11–12.

[Zel93] Alexander Zelikovsky. »An 11/6-approximation algorithm for the network Steiner problem«. In: *Algorithmica* 9 (Mai 1993), S. 463–470.

[Zel96] Alexander Zelikovsky. *Better Approximation Bounds for the Network and Euclidean Steiner Tree Problems*. Techn. Ber. 1996.

Abbildungsverzeichnis

Tabellenverzeichnis