Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik I
Effiziente Algorithmen und wissensbasierte Systeme

Diplomarbeit

Approximationsalgorithmen für das gerichtete Maximum-Leaf-Spanning-Tree-Problem

Nadine Schwartges

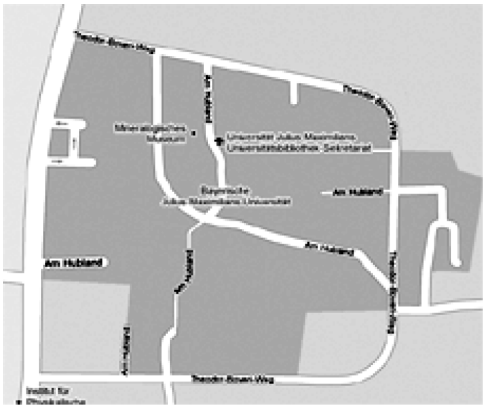
Abgabe: 28. März 2011

Betreuer:
Prof. Dr. Alexander Wolff
Dr. Joachim Spoerhase

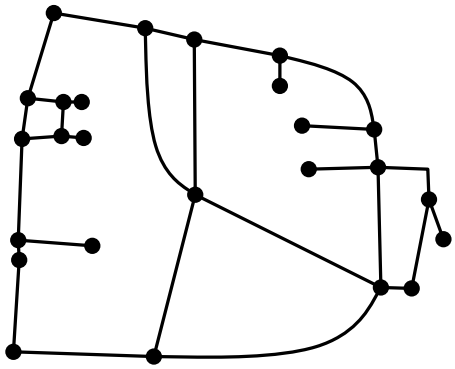
Inhaltsverzeichnis

1. Einführung

In der heutigen Zeit kommt jeder Mensch täglich, oft ohne es zu merken, mit Netzwerken in Berührung. Die meisten Leser werden nun an das Internet, Intranets oder Funknetze denken. Ebenso gibt es aber auch Netzwerke im nicht-technischen Bereich, etwa Verkehrsnetze oder Unternehmensnetzwerke. Abstrakt gesehen, sind alle Netzwerke gleich aufgebaut. Sie besitzen verknüpfende Stellen, die miteinander verbunden sind. Aufgrund dieser Gemeinsamkeit können alle Netzwerke über so genannte Graphen modelliert werden (vergleiche Abbildung 1.1). Ein Graph ist eine Menge von Knoten (angedeutet durch Punkte) und eine Menge von Kanten (Verbindungen zwischen Knoten). Ein Knoten stellt zum Beispiel einen Computer oder eine Straßenkreuzung dar. Die Kanten könnten zum Beispiel als Leitungen oder Straßenabschnitte interpretiert werden. Eine Kante deutet also in jedem Fall eine Beziehung zwischen zwei Knoten an.



Auszug aus der Würzburger Straßenkarte, Universität am Hubland

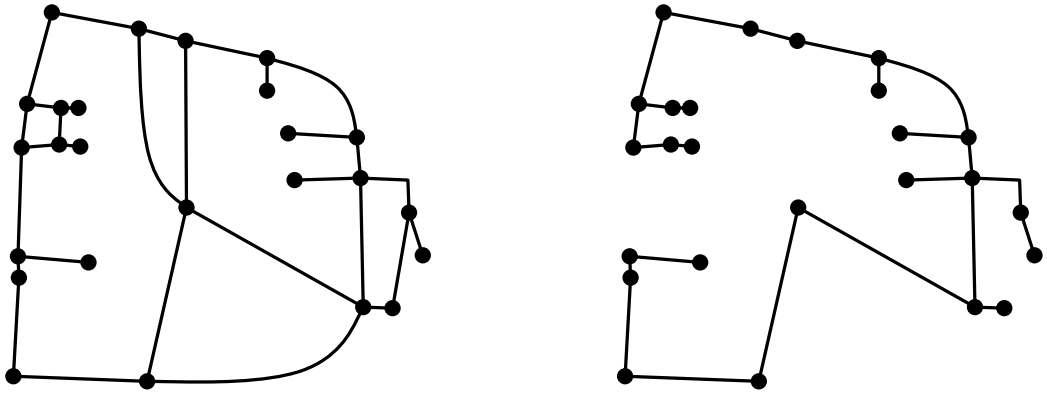


Straßenkarte als Graph wobei Kreuzungen als Knoten und Straßen als Kanten dargestellt sind; abgeschnittene Straßen werden vernachläßigt

Abb. 1.1.: Von der Straßenkarte zum Graphen.

In vielen praktischen Anwendungen besteht der Wunsch die Anzahl der Beziehungen gering zu halten. Für einen Graphen bedeutet das, dass er möglichst wenig Kanten enthält. Dennoch sollen alle Knoten in Verbindung stehen – direkt oder indirekt. Direkt bedeutet, dass zwei Knoten durch genau eine Kante verbunden sind. Indirekt bedeutet, dass zwei Knoten über einen Umweg über andere Knoten miteinander in Verbindung stehen. Abbildung 1.2(b) zeigt für das Netzwerk in Abbildung 1.2(a) eine Teilstruktur mit wenigen Beziehungen. Sie wird Spann-

baum genannt. Im Allgemeinen gibt es in einem Graphen sehr viele Spannbäume (vergleiche Abbildung 1.3).



(a) (b)

Abb. 1.2.: Vom Graphen zum Spannbaum.

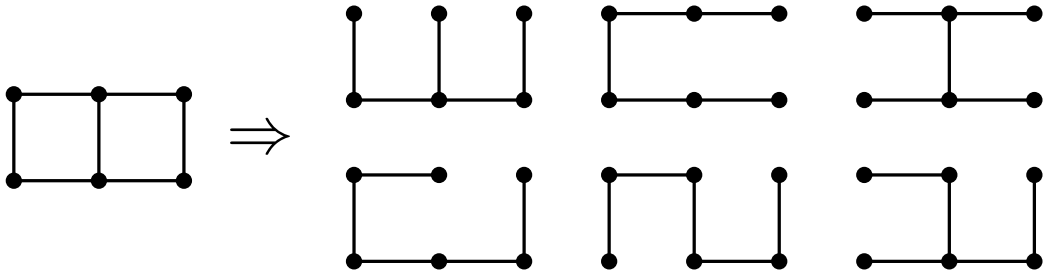


Abb. 1.3.: Ein Graph (links) und einige seiner Spannbäume (daneben stehend).

Der Grund für den Wunsch nach wenigen Beziehungen sind meistens Rationalisierungsmaßnahmen, das heißt, Senkung von Aufwand, Kosten und anderem.

In dieser Arbeit werden Methoden vorgestellt, die Spannbäume mit hoher Anzahl an Blättern ermitteln. Ein Blatt im graphentheoretischen Zusammenhang ist ein Knoten, welcher mit nur genau einer Kante verbunden ist. Im Gegensatz dazu stehen die inneren Knoten. Sie sind mit mindestens zwei Kanten verbunden. Die Maximierung der Anzahl der Blätter führt in vielen Fällen zu weiteren Vorteilen.

Im Folgenden werden drei praktische Probleme vorgestellt, die mit Hilfe von Graphen modelliert werden können. In allen drei Fällen ist die Lösung des Problems ein Spannbaum, wobei Spannbäume mit maximaler Blattanzahl wünschenswert sind.

Intranetdesign Betrachte ein Unternehmen, das ein Intranet aufbauen möchte. Es soll aus einem Server und mehreren Clients bestehen. Da die Räumlichkeiten schon bestehen und daher auch die Computerstandorte unveränderlich

sind, sind die Möglichkeiten die nötigen Leitungen zu legen begrenzt. Würde jeder Client direkt mit dem Server verbunden werden, könnten sehr lange Leitungen entstehen. Diese führen jedoch zu Leistungsabfall und Störungen.

Daher möchte das Unternehmen nur Leitungen bis zu einer bestimmten Länge verwenden. Da aber alle Computer mit dem Server verbunden werden sollen, ist ein Netzwerk nötig, welches aus zwei Computertypen besteht, nämlich aus Clients und Routern. Clients sind Computer, die ausschließlich empfangen können. Router hingegen sind Computer, die senden und empfangen können. Sie sind notwendig, wenn eine direkte Verbindung zwischen Server und Client (aufgrund der eingeschränkten Leitungslänge) nicht möglich ist.

Router sind jedoch aus verschiedenen Gründen teurer und aufwändiger als Clients. Sie benötigen zunächst eine besondere Hardwarekomponente, die gekauft und eingebaut werden muss. Des Weiteren ist eine spezielle Software nötig, welche Kosten bei der Anschaffung verursacht. Außerdem muss diese Software installiert und gewartet werden. Das Unternehmen strebt darum eine Lösung an, bei der möglichst wenig Router und damit viele Clients verwendet werden.

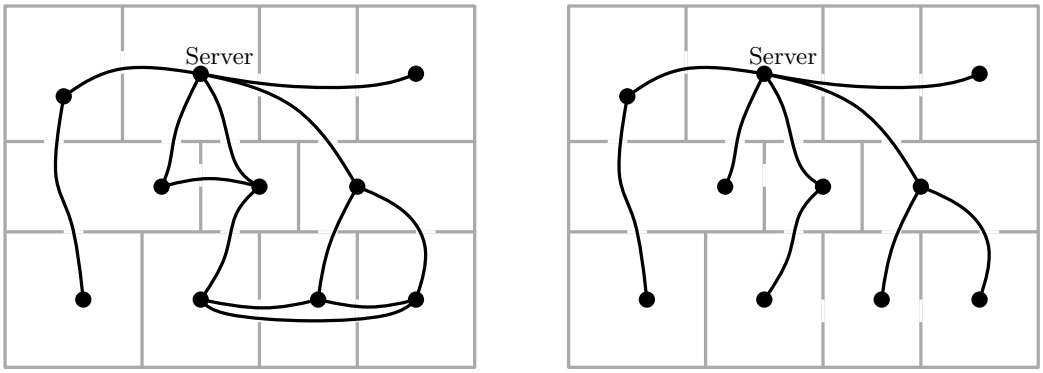


Abb. 1.4.: Modellierung des Intranetdesignproblems als Graph (links) mit zugehöriger Lösung (rechts).

Die linke Seite von Abbildung 1.4 zeigt wie dieses Szenario in der Praxis aussehen könnte. Sie zeigt einen Grundriss der Räumlichkeiten des Unternehmens inklusive der Stellen, an denen Leitungen durch die Wand verlegt werden können. Außerdem sind die Standorte des Servers (beschrifteter Knoten) und der Computer (übrige Knoten) eingezeichnet. Zuletzt wurden alle Kanten eingetragen, die einer Leitung entsprechen, welche die maximale Länge nicht überschreitet. Jeder Spannbaum in diesem Graphen liefert ein akzeptables Netzwerk, denn jeder Spannbaum minimiert die Anzahl an Leitungen. Jede weitere nicht genutzte Kante würde eine Verbindung zwischen Computer und Server unterbrechen. Hier wird jener Spannbaum gesucht, der die Anzahl der Blätter maximiert beziehungsweise die Anzahl der inneren Knoten minimiert. In diesem Szenario ist das nämlich gleichbedeutend mit

der Maximierung der Anzahl der Clients und der Minimierung der Anzahl der Router.

Die rechte Seite von Abbildung 1.4 zeigt eine Lösung des Beispiels.

Unternehmensnetzwerke Betrachte eine Reihe von Unternehmen, die sich entschlossen haben ein gemeinsames Projekt zu realisieren. Ein Unternehmen wird als Projektmanager bestimmt. Dieses delegiert Aufträge. Ebenso verteilt und sammelt es Informationen über den aktuellen Stand des Projekts, das heißt, es muss die Rückmeldungen der anderen Unternehmen verarbeiten können. Praktisch kann jedes Unternehmen mit jedem anderen kommunizieren. Dies ist jedoch nicht immer gewünscht, denn Kommunikation wirft mitunter Probleme auf. Oft werden Informationen fehlerhaft übermittelt. Beispielsweise werden Details hinzuerfunden oder weggelassen. Die sicherste Möglichkeit wäre natürlich, dass der Projektmanager mit jedem Unternehmen im Netzwerk persönlich in Kontakt tritt. Doch bei entsprechend großen Netzwerken mit komplexem Informationsaustausch würde der verursachte Aufwand zu hoch werden. Daher werden die Kommunikationswege begrenzt, das heißt, es soll nicht mehr jeder mit jedem kommunizieren.

Die begrenzten Kommunikationswege implizieren einen Graphen. Damit jeder die Informationen genau einmal zugetragen bekommt, wird in diesem Graphen ein Spannbaum gesucht. Da aber das Risiko der Informationsverfälschung besteht, soll es nur wenige Unternehmen geben, die Informationen weiterleiten. Also soll es viele Unternehmen geben, die Nachrichten nur empfangen. Für den Spannbaum bedeutet das, dass er möglichst viele Blätter besitzen soll.

Bahnhofsexpansion Betrachte einen Bahnhof, der aktuell eine Expansion plant. Diese soll im Bau neuer Abstellgleise bestehen. Aufgrund der Unebenheiten im Terrain können Schienen nicht willkürlich verlegt werden. Ein Ziel ist die Kosten für die Schienen gering zu halten. Ein anderes ist ein minimaler Rangieraufwand. Daher soll stets gewährleistet werden, dass ein Zug direkt auf seinen Stellplatz fahren oder von seinem Stellplatz wegfahren kann – ohne dass ein anderer Zug erst den Weg freigeben muss.

Auch dieses Problem ist als Graph modellierbar. Kanten stellen den Verlauf von Schienen dar, Blätter entsprechen Gleisabschlüssen und innere Knoten sind Weichen. Das Terrain beeinflusst die genaue Ausprägung des Graphen. Um die Kosten nun gering zu halten, wird in diesem Graphen ein Spannbaum gesucht. Dieser gewährleistet alle Gleisabschlüsse zu erreichen. Gleichzeitig werden keine unnötigen Schienen verlegt. Eine einfache Möglichkeit den Rangieraufwand zu minimieren ist jeden Zug vor einem Gleisabschluss abzustellen. Im Graphen wird also eine Kante, die in ein Blatt führt, als Abstellgleis verwendet. Alle weiteren Kanten dienen als Verbindungsstrecken. Auf ihnen wird kein Zug abgestellt. Damit aber möglichst viele Züge abge-

stellt werden können, muss die Anzahl der Gleisabschlüsse beziehungsweise der Blätter im Graphen maximiert werden.

Für diese Art von Problemen sind bereits viele Lösungsansätze bekannt. In dieser Arbeit wird eine naheliegende Verallgemeinerung des Problems behandelt. Bei dieser Verallgemeinerung dürfen manche Kanten nur in eine vorgegebene Richtung durchlaufen werden. Diese Kanten werden visuell mit Pfeilen dargestellt. Abbildung 1.5 zeigt einen Graph mit solchen Kanten. Er wird *gerichteter* Graph genannt. Die Abbildung zeigt außerdem einige mögliche Spannbäume. Sie sind ebenfalls gerichtet.

Für diesen verallgemeinerten Fall sind bislang nur deutlich weniger und schwächere Ergebnisse bekannt.

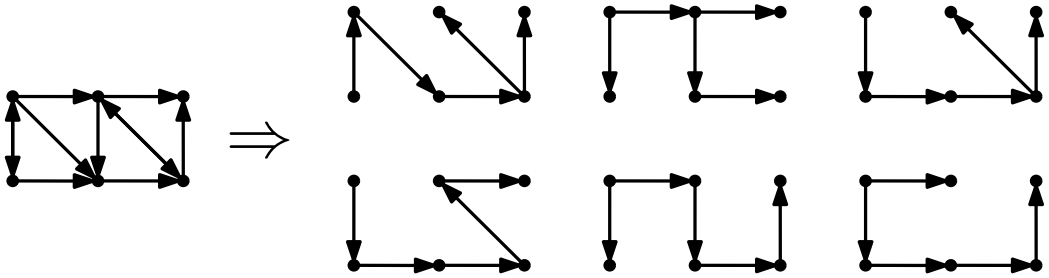


Abb. 1.5.: Ein gerichteter Graph (links) und einige seiner gerichteten Spannbäume (daneben stehend).

Die eben beschriebenen praktischen Anwendungen können so erweitert werden, dass ihre Lösung in einem gerichteten Graphen gefunden werden muss. Unternehmen könnten zum Beispiel verschiedene Ränge besitzen. Es könnte eine Regel geben, die besagt, dass Unternehmen mit niedrigerem Rang die Informationen nicht an privilegierte Unternehmen weiterleiten dürfen. Dann gibt es im Graphen Pfeile, die anzeigen, ob ein Unternehmen befugt ist Informationen an ein bestimmtes anderes Unternehmen weiter zu leiten.

Anstelle von Abstellgleisen können auch Entladestationen betrachtet werden. Aufgrund des Gefälles im Terrain könnten bestimmte Streckenabschnitte von vollbeladenen Zügen nicht passierbar sein. Nachdem sie entladen worden sind, können die Züge die Steigung jedoch überwinden. Dann muss der Zug vollbeladen abwärts und entladen aufwärts fahren. Auf diese Weise ist es gewährleistet, dass jeder Zug, trotz einer eventuellen Steigung, seine Entladestation erreichen und auch wieder verlassen kann. Im Graphen zeigen Pfeile an, welche Richtung nur entladen passierbar ist.

Das Problem in einem gerichteten Graphen einen Spannbaum mit maximaler Blattanzahl zu finden wird Rooted Maximum Leaf Outbranching (kurz RMLO-Problem) genannt. Aus gängigen komplexitätstheoretischen Annahmen folgt, dass es keine *effizienten* Algorithmen gibt, die das RMLO-Problem *exakt* lösen. Das bedeutet, dass es keinen Algorithmus gibt, der in angemessener Laufzeit

eine optimale Lösung findet. Aus diesem Grund werden in dieser Arbeit *approximative* Algorithmen für das RMLO-Problem untersucht. Solche Algorithmen finden zwar im Allgemeinen keine optimale Lösung, jedoch finden sie in angemessener Laufzeit eine gute Näherungslösung.

1.1. Mathematische Grundlagen

Nachfolgend werden alle in der Arbeit verwendeten Begriffe, die nicht von der Standarddefinition abweichen, kurz definiert. Für genauere Definitionen sei auf die Fachliteratur verwiesen.

Ungebräuchliche Begriffe, die diese Arbeit verwendet, werden bei ihrem ersten Vorkommen definiert. Wiederholt verwendete, ungebräuchlich Begriffe sind zudem im Anhang A (Glossar) aufgelistet.

Ein *Optimierungsproblem* P ist gegeben durch eine Menge zulässiger Instanzen $\mathcal{I}$. Außerdem ist für jede Instanz $I \in \mathcal{I}$ eine Menge zulässiger Lösungen gegeben. Die Zulässigkeit einer Instanz I sowie die Zulässigkeit einer Lösung für I ist in Polynomialzeit entscheidbar. Weiter ist eine Zielfunktion gegeben, die ebenfalls in Polynomialzeit berechenbar ist. Ziel ist es eine zulässige Lösung zu finden, die den Wert der Zielfunktion optimiert, das heißt, der Wert soll minimiert oder maximiert werden.

Viele dieser Optimierungsprobleme sind NP-schwer. (Für eine Einführung in das Konzept der NP-Schwere siehe Garey und Johnson [GJ79].) Die NP-Schwere eines Problems hat zur Folge, dass optimale Lösungen wahrscheinlich nicht in polynomieller Laufzeit ermittelbar sind. Darum wird zu Gunsten der Laufzeit auf Genauigkeit der Lösung verzichtet. Die Lösung erfolgt über so genannte *Approximationsalgorithmen*.

Ein Approximationsalgorithmus ist ein Algorithmus, der für ein Optimierungsproblem in Polynomialzeit eine zulässige, aber nicht notwendigerweise optimale Lösung liefert.

Ein *Faktor-α-Approximationsalgorithmus* (kurz *α-Approximationsalgorithmus*) ist ein Approximationsalgorithmus, der eine zulässige Lösung liefert, deren Zielfunktionswert höchstens um den Faktor α vom Optimum abweicht. Bei einem Maximierungsproblem bedeutet das, dass mit dem Approximationsalgorithmus für jede Instanz I des betrachteten Problems eine Lösung gefunden wird, die garantiert mindestens $1/\alpha$-mal so gut wie das Optimum für I ist. Bei einem Minimierungsproblem wird für jede Instanz I des betrachteten Problems eine Lösung gefunden, die garantiert höchstens α-mal so schlecht wie eine optimale Lösung für I ist.

Der Faktor α wird auch *Güte* genannt.

Handelt es sich bei der Güte um eine *scharfe Schranke*, so bedeutet das, dass Eingabeinstanzen gefunden werden können, die genau diese Güte aufweisen.

Ein *Polynomialzeitapproximationschema* (auch *polynomial time approximation scheme*, kurz *PTAS*) für ein bestimmtes Optimierungsproblem ist ein Approximationsalgorithmus mit besonderen Eigenschaften. Er besitzt für jedes $\epsilon > 0$ eine Güte

von $(1+\epsilon)$. Außerdem ist die Laufzeit eines PTAS für jedes feste ϵ polynomiell in der Größe der Eingabeinstanz.

Approximationsalgorithmen basieren häufig auf einem *Greedy-Ansatz*. Ein Greedy-Algorithmus startet mit einer leeren Lösung. Er wählt in jedem Schritt immer die aktuell beste Möglichkeit und fügt sie zur Lösung hinzu. Es wird kein Schritt revidiert.

Ebenso typisch sind Approximationsalgorithmen, die auf einer *lokalen Suche* basieren. Eine lokale Suche startet mit einer beliebigen Lösung. Diese wird solange modifiziert, bis ein lokales Optimum erreicht ist. Das bedeutet, dass eine erneute Modifikation keine Verbesserung der Lösung mehr bewirkt.

Bei einem *Entscheidungsproblem* soll entschieden werden, ob eine Eingabeinstanz eine gewünschte Eigenschaft erfüllt.

Parametrisierbare Probleme sind Entscheidungsprobleme bei denen jeder zulässigen Instanz I ein Parameter $k(I)$ (kurz k) zugeordnet wird.

Ein parametrisierbares Problem heißt *fest-parameter-berechenbar* (auch *fixed parameter tractable*, kurz *FPT)*, wenn es einen Algorithmus gibt, der das Problem in Laufzeit $\mathcal{O}(f(k) \cdot p(|I|))$ löst, wobei $|I|$ die Länge der Eingabeinstanz I, f eine berechenbare, von I unabhängige Funktion und p ein beliebiges Polynom ist.

Eine Möglichkeit zur Lösung eines fest-parameter-berechenbaren Problems (I,k) ist die Verwendung eines *Kerns* (I',k). Genauer gesagt wird (I,k) auf (I',k) abgebildet, indem die Größe der Eingabeinstanz verringert ("reduziert") wird. Die Größe der reduzierten Instanz hängt nur noch vom Parameter k ab, das heißt, $|I'| = f(k)$. Hierbei ist f eine berechenbare Funktion. Die Instanz I' muss in Polynomialzeit berechenbar sein. Außerdem muss der Kern (I',k) die Frage nach der Eigenschaftserfüllung genau dann mit wahr beantworten, wenn das Eingabeproblem (I,k) die Eigenschaft erfüllt. Die Lösung der Instanz (I',k) erlaubt schließlich Rückschlüsse auf die Lösung des Problems (I,k). Das Problem (I,k) ist damit insgesamt in $\mathcal{O}(f'(k)+|I|^c)$ mit c konstant und f' geeignete berechenbare Funktion lösbar.

Ein *ungerichteter Graph* G ist ein Paar (V,E), wobei $V \neq \emptyset$ eine endliche Menge so genannter Knoten und $E \subseteq \{\{u,v\} \mid u,v \in V, u \neq v\}$ eine endliche Menge so genannter Kanten ist. Ein Graph, der Pfeile (gerichtete Kanten) enthält, heißt auch *gerichteter Graph* oder *Digraph*. Seine Kanten sind geordnete Paare $e = (u,v)$. In den folgenden Kapiteln dieser Arbeit werden jedoch sowohl gerichtete als auch ungerichtete Kanten vereinfacht mit runden Klammern notiert. Außerdem wird bei gegebenen Graphen G die Knotenmenge auch mit $V(G)$ und die Kantenmenge analog mit $E(G)$ notiert.

Der *Grad* $\deg(v)$ eines Knoten v gibt an, wieviele Kanten zu diesem Knoten v *inzident* sind, das heißt, wieviele Kanten mit diesem Knoten in direkter Verbindung stehen. Für gerichtete Kanten kann zwischen dem *Eingangsgrad* $\deg^-(v)$ der in den Knoten v eingehenden und dem *Ausgangsgrad* $\deg^+(v)$ der ausgehenden Kanten eines Knoten v unterschieden werden. Es gilt $\deg(v) = \deg^-(v) + \deg^+(v)$.

Ein *gradbeschränkter Graph* ist ein Graph, dessen Grade durch eine Konstante nach oben beschränkt sind.

Die *Nachbarschaft* $N(v)$ eines Knoten v im Graphen $G = (V, E)$ ist die Menge aller Knoten, die durch eine Kante mit v verbunden sind, in Zeichen $N(v) = \{u \in V \mid \{v, u\} \in E\}$. Gilt $\{u, v\} \in E$ werden die zwei Knoten u und v *benachbart* genannt. Falls $(u, v) \in E$, also im gerichteten Fall, heißt der Knoten u *Vorgänger* von v und der Knoten v heißt *Nachfolger* von u. Die Menge aller Vorgänger eines Knoten v wird mit $N^-(v)$ notiert. Analog ist $N^+(v)$ die Menge aller Nachfolger eines Knoten v.

Ein *einfacher Pfad* von v_i nach v_j ist eine Folge von Knoten $v_i, v_{i+1}, \ldots, v_{j-1}, v_j$, sodass zwei aufeinander folgende Knoten v_k, v_{k+1} für $k = i, \ldots, j-1$ jeweils durch eine Kante (v_k, v_{k+1}) verbunden sind. Knoten des Pfades sind paarweise verschieden. In dieser Arbeit werden einfache Pfade kurz als Pfade bezeichnet.

Im Graph G heißt ein Knoten u *erreichbar* vom Knoten v, falls ein Pfad von v nach u in G existiert.

Die *Wurzel* r eines Graphen $G = (V, E)$ ist ein ausgezeichneter Knoten, der alle Knoten $v \in V \setminus \{r\}$ erreicht und Eingangsgrad $\deg^-(r) = 0$ besitzt. Es existieren also keine in r eingehenden Kanten. Damit kann die Wurzel insbesondere nicht mit einer *bidirektionalen Kante* inzidieren. Eine bidirektionale Kante $\{u, v\}$ ist die Vereinfachung der zwei *unidirektionalen* (*einfachen*) Kanten (u, v) und (v, u). Eine bidirektionale Kante ist also in zwei Richtungen gerichtet.

Besitzt ein gerichteter Graph eine Wurzel, dann wird dieser auch als *Wurzelgraph* oder *gewurzelter Graph* bezeichnet.

Eine *Senke* ist ein Knoten v in einem gerichteten Graphen, der keinen Nachfolger besitzt, das heißt, $\deg^+(v) = 0$.

Ein *Blatt* ist ein Knoten b mit Grad $\deg^+(b) = 0$ und $\deg^-(b) = 1$. Im ungerichteten Fall wird vereinfacht Grad $\deg(b) = 1$ gefordert.

Ein Knoten heißt *innerer Knoten*, wenn er weder Blatt noch Wurzel ist, falls diese existiert.

Ein *isolierter Knoten* x besitzt Knotengrad $\deg(x) = 0$ beziehungsweise $\deg^+(x) = \deg^-(x) = 0$.

Ein *Kreis* ist ein Pfad, dessen Start- und Endknoten gleich ist, das heißt, ein Pfad der Form $(u_1, u_2, \ldots, u_k, u_1)$ mit $k \geq 3$ bei ungerichteten und $k \geq 2$ bei gerichteten Graphen. Ein ungerichteter Graph ohne Kreis heißt *azyklisch* oder *Wald*. Ein gerichteter Graph ohne Kreis heißt azyklisch. Ist ein Graph gerichtet und der zugehörige ungerichtete Graph azyklisch, so heißt der Graph auch *Wald*.

Ein ungerichteter Graph $G = (V, E)$ heißt *zusammenhängend*, wenn von einem Knoten $v \in V$ alle anderen Knoten $u \in V \setminus \{v\}$ erreichbar sind. Im gerichteten Fall werden *starker* und *schwacher Zusammenhang* unterschieden. Ein Graph $G = (V, E)$ heißt stark zusammenhängend, wenn für jeden Knoten $v \in V$ jeder andere Knoten $u \in V \setminus \{v\}$ erreichbar ist. Ein Graph G heißt schwach zusammenhängend, wenn sein zugehöriger ungerichteter Graph zusammenhängend ist. Verkürzend wird in diesem Fall in der Regel nur vom Zusammenhang gesprochen.

Ein ungerichteter Graph G heißt *Baum*, wenn der Graph G azyklisch und zusammenhängend ist. Ein gerichteter Graph $G = (V, E)$ heißt Baum, wenn er eine Wurzel r besitzt und außerdem jeder Knoten $v \in V \setminus \{r\}$ nur genau einen Vorfahren,

das heißt, Eingangsgrad $\deg^-(v) = 1$, besitzt.

In einem ungerichteten Baum $T = (V, E)$ existiert für jedes Knotenpaar v_i, v_j mit $v_i, v_j \in V$ genau ein Pfad $P(v_i, v_j)$, wobei v_i und v_j die Endpunkte des Pfades sind. Im gerichteten Baum $T = (V, E)$ hingegen existiert für jedes Knotenpaar v_i, v_j mit $v_i, v_j \in V$ höchstens ein Pfad $P(v_i, v_j)$, wobei der Pfad am Knoten v_i startet und am Knoten v_j endet.

Bei einem *Binärbaum* handelt es sich um einen Baum $T = (V, E)$ mit Wurzel r, bei dem jeder Knoten $v \in V$ maximal zwei ausgehende Kanten und jeder Knoten $v \in V \setminus \{r\}$ genau eine eingehende Kante besitzt. Die Kanten können gerichtet oder ungerichtet sein. Ein Knoten v liegt in *Ebene* i, wenn der Knoten v Endpunkt des Pfades $P(r, v)$ ist, der genau i Kanten verwendet. Die Wurzel liegt also in Ebene 0.

Ein *vollständiger Binärbaum* besitzt in Ebene $i + 1$ genau doppelt soviele Knoten wie in Ebene i. Alle Blätter liegen in derselben Ebene. Sie sei als Ebene k bezeichnet. Ein vollständiger Binärbaum besitzt genau $2^{k+1} - 1$ Knoten und genau 2^k Blätter.

Ein *Teilgraph* $G' = (V', E')$ eines Graphen $G = (V, E)$ enthält nur Teilmengen der Knoten- und Kantenmenge von G, also $V' \subseteq V$ und $E' \subseteq E$. Handelt es sich beim entstandenen Teilgraphen um einen Wald, so wird der Teilgraph auch als *Teilwald* bezeichnet.

Ein *Spannbaum* oder *aufspannender Baum* ist ein Teilgraph $T = (V', E')$ von $G = (V, E)$ mit $V' = V$ und $E' \subseteq E$, welcher die Baumeigenschaften erfüllt.

Jeder maximale zusammenhängende Teilgraph eines ungerichteten Graphen heißt *Zusammenhangskomponente*. Im gerichteten Fall werden *starke* und *schwache Zusammenhangskomponenten* unterschieden. Ihre Unterscheidung wird analog dem starken und schwachen Zusammenhang getroffen. Da in dieser Arbeit ausschließlich schwache Zusammenhangskomponenten betrachtet werden, werden sie verkürzend als Zusammenhangskomponente bezeichnet.

Ein *Gelenkpunkt* (auch *cut vertex*) ist ein Knoten, dessen Entfernung aus Graph G zur Erhöhung der Anzahl der Zusammenhangskomponenten von G führt.

Ein *induzierter Teilgraph* $G[V'] = (V', E')$ eines Graphs $G = (V, E)$ ergibt sich durch Löschung der Knoten aus $V \setminus V'$ sowie der zu diesen Knoten inzidenten Kanten aus G.

Eine *topologische Sortierung* eines gerichteten, kreisfreien Graphen $G = (V, E)$ ist eine Abbildung der Knotenmenge $f : V \to \{1, 2, \ldots, |V|\}$, sodass für jede Kante $(u, v) \in E$ $f(u) < f(v)$ gilt.

Eine *dominierende Menge* D ist eine Menge von Knoten in einem ungerichteten Graphen $G = (V, E)$, sodass jeder Knoten $v \in V \setminus D$ einen Nachbarn in der dominierenden Menge $D \subseteq V$ besitzt. Eine *kleinste dominierende Menge* ist eine dominierende Menge, welche $|D|$ minimiert.

Eine dominierende Menge in einem gerichteten Graphen G dominiert die Knoten des zu G zugehörigen ungerichteten Graphen.

Eine *zusammenhängende dominierende Menge* D in einem Graphen G ist eine dominierende Menge, sodass $G[D]$ zusammenhängend ist.

Das Problem Steiner Tree (kurz ST-Problem) ist ein Spannbaumproblem. Gegeben sei ein ungerichteter, zusammenhängender, kantengewichteter Graph $G = (V, E)$ und eine Knotenmenge $T \subseteq V$, genannt Terminale. Gesucht ist ein Teilgraph von G, der alle Terminale erreicht und die Summe der Kantengewichte der genutzten Kanten minimiert. Dieser Teilgraph wird *Steinerbaum* genannt.

Das ST-Problem in gerichteten Graphen ist analog definiert. Die Wurzel des Steinerbaums ist durch einen ausgezeichneten Knoten der Eingabeinstanz vorgegeben.

1.2. Problemdefinition

Definition 1 (gerichteter Spannbaum)**.** Ein *gerichteter Spannbaum* in einem gerichteten Graphen $G = (V, E)$ ist ein Spannbaum mit einer beliebigen Wurzel $r \in V$ in G, bei dem jeder Knoten $v \in V$ von r aus erreicht werden kann.

Definition 2 (Outbranching)**.** In einem gerichteten Graphen $G = (V, E)$ mit ausgezeichnetem Knoten $r \in V$ ist ein *Outbranching* ein gerichteter Spannbaum mit Wurzel r.

Im weiteren Verlauf der Arbeit wird auf die explizite Angabe der Wurzel r verzichtet, sofern deren Existenz aus dem Kontext klar ersichtlich ist.

Problem (Rooted Maximum Leaf Outbranching)**.** Gegeben sei ein gerichteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $r \in V$, finde unter allen Outbranchings mit Wurzel r eines mit maximaler Anzahl an Blättern. Solch ein optimales Outbranching von G mit Wurzel r heißt auch RMLO. Das Problem wird kurz als RMLO-Problem bezeichnet.

Das dazugehörige Entscheidungsproblem ist NP-vollständig [BG09].

1.3. Frühere Ergebnisse

Der ungerichtete Fall des RMLO-Problems wurde häufig untersucht. Unter anderem stellten Lu und Ravi [LR92] einen Approximationsalgorithmus vor, welcher auf dem Prinzip der lokalen Suche basiert und von einem Parameter k beeinflusst wird. Für $k = 1$ haben Lu und Ravi eine Güte von 5, für $k = 2$ eine Güte von 3 bewiesen. Solis-Oba [Sol98] verbesserte dieses Ergebnis, indem er einen Greedy-Algorithmus mit Güte 2 und linearer Laufzeit vorstellte.

Bang-Jensen und Gutin [BG09] haben bewiesen, dass das RMLO-Problem NP-schwer ist.

Daligault und Thomassé [DT09] haben einen 92-Approximationsalgorithmus für das (gerichtete) RMLO-Problem beschrieben. Dieser ist besser als der einzige zuvor bekannte Approximationsalgorithmus für das Problem Maximum Leaf Spanning Arborescence (kurz MLSA-Problem), wobei beim MLSA-Problems im Unterschied zum RMLO-Problem kein ausgezeichneter Knoten als Wurzel vorgeben

ist. Ansonsten sind die Probleme identisch. Der erwähnte, schlechtere Algorithmus stammt von Drescher und Vetta [DV10]. Seine Güte beträgt $\mathcal{O}(\sqrt{OPT})$, wobei OPT die maximale Blattanzahl ist.

Beim MLSA-Problem wurde bislang vorrangig die fest-Parameter-Berechenbarkeit und exakte Exponentialzeitalgorithmen untersucht. Die Existenz eines FPT-Algorithmus für bestimmte Graphklassen wurde von Alon et al. [AFG+07] gezeigt. Bonsma und Dorn [BD07] verallgemeinerten den Beweis von Alon et al. auf beliebige gerichtete Graphen. Daraufhin wurde eine Reihe von fest-Parameter-Algorithmen mit immer besseren Laufzeiten angegeben. Das momentan besten Resultat ist von Binkele-Raible und Fernau [BF10]. Sie haben einen exakten Algorithmus mit der Laufzeit $\mathcal{O}^*(1,9043^n)$ vorgestellt, wobei n die Anzahl der Knoten der Eingabeinstanz ist und die $\mathcal{O}^*$-Notation polynomielle Faktoren der $\mathcal{O}$-Notation ignoriert. Der benötigte Speicherplatz ist polynomiell begrenzt. Sie zeigten auch, dass mit exponentiell großem Speicherplatz eine Verbesserung auf $\mathcal{O}^*(1,8139^n)$ erreicht werden kann.

Aufgrund der Komplexität des Problems wurden auch spezielle Graphklassen untersucht. Sie führten zu besseren Ergebnissen. Dorn et al. [DFL+10] erreichten auf planaren Graphen eine Laufzeit von $2^{\mathcal{O}(\sqrt{k})}+n^{\mathcal{O}(1)}$, wobei n die Anzahl der Knoten der Eingabeinstanz und k der Parameter ist. Für azyklische Graphen zeigten Alon et al. [AFG+09] eine Laufzeit von $2^{\mathcal{O}(k \log k)} \cdot n^{\mathcal{O}(1)}$. Dass azyklische Graphen einen linearen Kern besitzen, wurde von Daligault et al. [DGKY10] bewiesen.

Allerdings sind keine Approximationsfaktoren für spezielle Graphklassen bekannt, die besser sind als die Approximationsfaktoren des allgemeinen RMLO-Problems.

1.4. Verwandte Probleme

Das Problem Maximum Leaf Spanning Tree (kurz MLST-Problem) ist ein Spezialfall des RMLO-Problems. In einem gegebenen ungerichteten Graphen soll ein Spannbaum mit maximaler Anzahl an Blättern gefunden werden. Das bedeutet, dass die beiden Anwendungen sich nur in der Eigenschaft des Eingabegraphen unterscheiden. Dieser ist beim RMLO-Problem gerichtet und beim MLST-Problem ungerichtet.

Eine Verallgemeinerung des RMLO-Problems ist das Problem Maximum Leaf Spanning Arborescence (kurz MLSA-Problem). Bei diesem wird ein gerichteter Spannbaum in einem gegebenen gerichteten Graphen gesucht. Der Unterschied liegt also gerade darin, dass beim MLSA-Problem die Wurzel nicht fest vorgegeben ist.

Ein weiteres Problem ist Rooted Maximum Leaf Out-Tree. Ein Out-Tree ist ein Wurzelbaum mit maximaler Blattanzahl, der im Gegensatz zum Outbranching nicht notwendigerweise aufspannend ist. Jedoch ist aus einem Out-Tree ein Outbranching mit gleicher Blattanzahl konstruierbar. Dafür werden die nicht enthaltenen Knoten über nicht enthaltene Kanten hinzugefügt. Mit jeder neuen Kante wird ein Blatt zerstört und ein neues erstellt [KLR08].

Statt die maximale Anzahl an Blättern zu bestimmen, kann es auch ein Ziel sein diese zu minimieren beziehungsweise die Anzahl der inneren Knoten zu maximieren. Dieses Problem wird als Minimum Leaf Outbranching bezeichnet.

1.5. Forschungsziel

Ziel dieser Arbeit ist es Methoden zu entwickeln, die das RMLO-Problem (vergleiche Problemdefinition auf Seite 10) approximativ lösen.

Zunächst soll untersucht werden inwiefern sich bekannte Approximationsalgorithmen mit konstanter Güte, die den ungerichteten Fall des RMLO-Problems (namentlich MLST-Problem) lösen, auf den gerichteten Fall übertragen lassen. Dieser Ansatz ist naheliegend, da für das MLST-Problem zahlreiche starke Ergebnisse bekannt sind. Zweitens sollen Verfahren entworfen werden, die für spezielle Graphklassen besser sind als der noch nicht ganz zufriedenstellende 92-Approximationsalgorithmus für das allgemeine RMLO-Problem von Daligault und Thomassé [DT09].

Die entworfenen Algorithmen sollen mit mathematischen Analysemethoden untersucht werden, das heißt, ihre Approximationsfaktoren sollen mathematisch bewiesen werden. Es sollen keine experimentellen Evaluierungen durchgeführt werden.

1.6. Überblick

In Kapitel 2 werden zwei bekannte Ansätze zur Lösung des RMLO-Problems (vergleiche Problemdefinition auf Seite 10) in ihren Grundzügen beschrieben. Es handelt sich um einen fest-parameter-berechenbaren Ansatz und einen 92-Approximationsalgorithmus von Daligault und Thomassé [DT09]. In Kapitel 3 folgt die Vorstellung von zwei bekannten Algorithmen, welche den ungerichteten Fall des RMLO-Problems approximativ lösen. Es werden einige Beispiele angeführt, die belegen, dass es nicht ohne Weiteres möglich ist die Algorithmen für den ungerichteten Fall auf den gerichteten Fall zu übertragen. Die Beispiele zeigen, dass die intuitiv übertragenen Algorithmen keine festen Approximationsfaktoren besitzen beziehungsweise nicht immer eine zulässige Lösung liefern. Zusätzlich wird eine Idee skizziert, die Hinweise zur Lösung des allgemeinen RMLO-Problems gibt.

Danach werden azyklische Graphen fokussiert. Zunächst wird in Abschnitt 4.1 und 4.2 gezeigt, dass es selbst in diesem Spezialfall kein PTAS gibt. Daher gibt es eine konstante untere Schranke für den Approximationsfaktor des azyklischen RMLO-Problems. Auf der positiven Seite werden Approximationsalgorithmen mit konstanter Güte entwickelt. Diese Güte ist substantiell besser als der 92 Faktor von Daligault und Thomassé. Konkret handelt es sich dabei um einen 4-Approximationsalgorithmus und einen 2-Approximationsalgorithmus. Sie werden in Abschnitt 4.3 und 4.4 vorgestellt und stellen das Hauptergebnis dieser Arbeit dar.

2. Ansätze nach Daligault und Thomassé

Der beste bisher bekannte Approximationsalgorithmus, der ein Outbranching mit vielen Blättern in allgemeinen Graphen erstellt, wurde von Daligault und Thomassé [DT09] vorgestellt. Dieser hat eine Güte von 92. Er verbessert den einzigen zuvor bekannten Approximationsalgorithmus, welcher eine Güte von $\mathcal{O}(\sqrt{OPT})$ besitzt [DV10].

2.1. Fest-Parameter-Berechenbarkeit

Daligault und Thomassé [DT09] untersuchen zuerst das parametrisierte Problem. Dieses handelt von der Frage, ob in einem gegebenen gerichteten Graphen mit vorgegebener Wurzel ein Outbranching mit mindestens k Blättern existiert. Dabei ist die Anzahl k der Blätter Teil der Eingabe und wird als Parameter verwendet.

Daligault und Thomassé beweisen in ihrer Arbeit zunächst zwei Schranken. Diese sind in den nachfolgenden Lemmata 2.1 und 2.2 wiedergegeben. Zum Verständnis der Lemmata werden zunächst zwei Begriffe eingeführt.

Definition 3 (In-/Out-$d(^*)$-Knoten)**.** Ein Knoten wird *In-d-Knoten* genannt, wenn er genau d eingehende Kanten besitzt. Hingegen wird ein Knoten, der mindestens d eingehende Kanten besitzt, als *In-d*-Knoten* bezeichnet. *Out-d-Knoten* beziehungsweise *Out-d*-Knoten* sind analog für ausgehende Kanten definiert.

Ein Wurzelgraph $G = (V, E)$ wird hier als *zweifach zusammenhängend* bezeichnet, wenn es für jeden Knoten $v \in V \setminus (N^+(r) \cup \{r\})$ mindestens zwei knotendisjunkte Pfade gibt, die in der Wurzel r starten und im Knoten v enden. Es sei darauf hingewiesen, dass diese Definition nicht der Standarddefinition des zweifachen Zusammmenhangs entspricht.

Lemma 2.1 (Daligault und Thomassé [DT09])**.** *Sei G ein zweifach zusammenhängender Wurzelgraph mit ℓ In-3*-Knoten, dann hat G ein Outbranching mit wenigstens $\ell/6$ Blättern.*

Lemma 2.2 (Daligault und Thomassé [DT09])**.** *Sei G ein zweifach zusammenhängender Wurzelgraph mit ℓ' Knoten, die mindestens eine unidirektionale eingehende Kante besitzen, dann hat G ein Outbranching mit wenigstens $\ell'/24$ Blättern.*

Ausgehend von Lemma 2.1 und 2.2 wird in der Arbeit von Daligault und Thomassé der Begriff der *speziellen Knoten* eingeführt. Dies sind alle In-3*-Knoten sowie Knoten, die mindestens eine unidirektionale eingehende Kante besitzen. Daraus ergibt sich eine neue Schranke. Diese ist in Lemma 2.3 dargestellt.

Lemma 2.3 (Daligault und Thomassé [DT09])**.** *Ist G ein zweifach zusammenhängender Wurzelgraph mit $30k - 1$ $(= 6k + 24k - 1)$ speziellen Knoten, dann besitzt G ein Outbranching mit mindestens k Blättern.*

Weiter stellen Daligault und Thomassé parameterunabhängige Reduktionsregeln vor. Diese transformieren die Eingabeinstanz G in einen kleineren, äquivalenten Graphen G'. Der Graph G' ist zweifach zusammenhängend. Außerdem besitzen der Eingabegraph G und die reduzierte Instanz G' dieselbe Anzahl an Blättern. Das heißt, dass ein allgemeiner Wurzelgraph G auf einen Problemkern reduziert wird. Aufgrund des zweifachen Zusammenhangs wird die oben angeführte Schranke aus Lemma 2.3 anwendbar. Zur Entscheidungsfindung genügt damit eine Prüfung, ob der reduzierte Graph mindestens $30k - 1$ spezielle Knoten besitzt.

Diese Prüfung setzt jedoch Wissen über die Anzahl der speziellen Knoten in der reduzierten Instanz G' voraus. Daligault und Thomassé schätzen die Anzahl der speziellen Knoten in der Instanz G' mit Hilfe der Anzahl $|V(G')|$ aller Knoten ab. Dazu zeigen sie, dass die Instanz G' insgesamt höchstens $(3k - 3)(30k - 2)$ Knoten besitzt, falls sie weniger als $30k - 1$ spezielle Knoten besitzt. Ist dies der Fall, dann kann die Frage nach einem Outbranching mit mindestens k Blättern mit einem Zeitaufwand beantwortet werden, der nur von k abhängt. Dieser kann jedoch exponentiell sein.

Umgekehrt bedeutet dies aber auch, dass eine reduzierten Instanz G' mit einer Größe von mindestens $(3k - 2)(30k - 2)$ Knoten mindestens $30k - 1$ spezielle Knoten besitzt. Dann kann in der Instanz G' ein Outbranching mit mindestens k Blätter gefunden werden. Diese Eigenschaft überträgt sich auf die unreduzierte Eingabeinstanz G.

Demnach handelt es sich beim RMLO-Problem um ein fest-parameter-berechenbares Problem.

2.2. Approximierbarkeit

Die von Daligault und Thomassé entwickelten Reduktionsregeln (vergleiche Abschnitt 2.1) sind unabhängig vom Parameter k und *umkehrbar*. Umkehrbarkeit bedeutet in diesem Fall, dass aus jedem Outbranching T' von der reduzierten Instanz G' ein Outbranching T von der Eingabeinstanz G gewonnen werden kann. Dabei ist die Anzahl der Blätter in T mindestens so hoch wie die Anzahl der Blätter in T'. Aufgrund der beschriebenen Umkehrbarkeit und des konstruktiven Vorgehens in den Beweisen ist es möglich nicht nur das Entscheidungsproblem zu lösen, sondern auch ein approximierendes Outbranching zu ermitteln. Dieses besitzt mindestens $1/92$ der Blätter, die ein Outbranching mit maximaler Blattanzahl besitzt.

Der Approximationsalgorithmus von Daligault und Thomassé verwendet im Gegensatz zum parametrisierten Ansatz nur eine Reduktionsregel. Diese erstellt aus einem beliebigen Wurzelgraphen G einen zu G äquivalenten zweifach zusammenhängenden Graphen G'.

Aufgrund des zweifachen Zusammenhangs stellen schon die konstruktiven Beweise zu Lemma 2.1 und 2.2 Methoden zur Erstellung eines Outbranchings bereit. Daligault und Thomassé stellen noch eine dritte Möglichkeit vor. Diese drei Konstruktionen dienen als Beweisgrundlage für eine obere und eine untere Schranke für die Blattanzahl im ermittelten Outbranching. Aus diesen beiden Schranken kann der Approximationsfaktor 92 abgeleitet werden. Daligault und Thomassé schlagen vor, mit den drei vorgestellten Methoden drei Outbranchings zu erstellen und den mit der höchsten Blattanzahl als Ausgabe zu wählen.

3. Lösungsansätze für allgemeine Graphen

Ein naheliegender Ansatz zur Entwicklung eines Approximationsalgorithmus zur Lösung des RMLO-Problems besteht darin die Algorithmen für den ungerichteten Fall des RMLO-Problems, also die Algorithmen für das MLST-Problem, auf gerichtete Graphen zu übertragen.

Unter den Arbeiten über das umfassend untersuchte MLST-Problem befinden sich unter anderem ein Approximationsalgorithmus von Lu und Ravi [LR92], welcher je nach Parameterwahl Güte 5 oder 3 besitzt, und der von Solis-Oba [Sol98] vorgestellte 2-Approximationsalgorithmus. Die Algorithmen basieren auf einer lokalen Suche beziehungsweise einem Greedy-Verfahren.

In diesem Kapitel soll nun untersucht werden, in wie weit sich die Verfahren für die ungerichteten Graphen auf gerichtete Graphen übertragen lassen. Abschnitt 3.1 und 3.2 behandeln die lokale Suche, Abschnitt 3.3 und 3.4 das Greedy-Verfahren.

3.1. Lokale Suche in ungerichteten Graphen

Der Approximationsalgorithmus von Lu und Ravi [LR92] ermittelt auf einem zusammenhängenden und ungerichteten Graphen einen Spannbaum mit hoher Blattanzahl mit Hilfe der so genannten *k-Change-Methode* (auch *k-swap* oder *Kantentauschverfahren)*, wobei $k \geq 1$ fest ist.

Der Algorithmus ist in Algorithmus 1 dargestellt. Er startet mit der Erstellung eines beliebigen Spannbaums T im Eingabegraphen G. Dann wird ein k-Change durchgeführt, das heißt, es werden k Kanten des Baumes T und k Kanten des Graphen G, welche nicht in T verwendet werden, so gewählt, dass bei einem Tausch die Baumeigenschaften erhalten bleiben. Der Tausch wird nur dann durchgeführt, wenn die Anzahl der Blätter nach dem Tausch echt höher ist als vor dem Tausch. Diese Tauschoperation wird solange durchgeführt bis eine erneute Anwendung des k-Change keine Verbesserung im Hinblick auf die Anzahl der Blätter mehr bewirkt. Das Ergebnis ist lokal optimal.

Lu und Ravi haben bewiesen, dass der Algorithmus für $k = 1$ eine 5-Approximation und für $k = 2$ eine 3-Approximation liefert. Für $k > 2$ sind keine besseren Approximationsfaktoren bekannt. Die Laufzeit beträgt $\mathcal{O}(n^{3k+1})$, wobei n die Anzahl der Knoten im Eingabegraphen ist. Das heißt, mit wachsendem k steigt die Laufzeit.

Algorithmus 1: k-Change-Algorithmus(G, k)

(1) **Input** : zusammenhängender, ungerichteter Graph G, Parameter $k \geq 1$
(2) **Output** : Spannbaum T mit hoher Blattanzahl
(3) finde einen Spannbaum T in G
(4) **while** *ein verbessernder k-Change C existiert* **do**
(5) aktualisiere T durch Anwendung von C
(6) **end while**
(7) **return** T

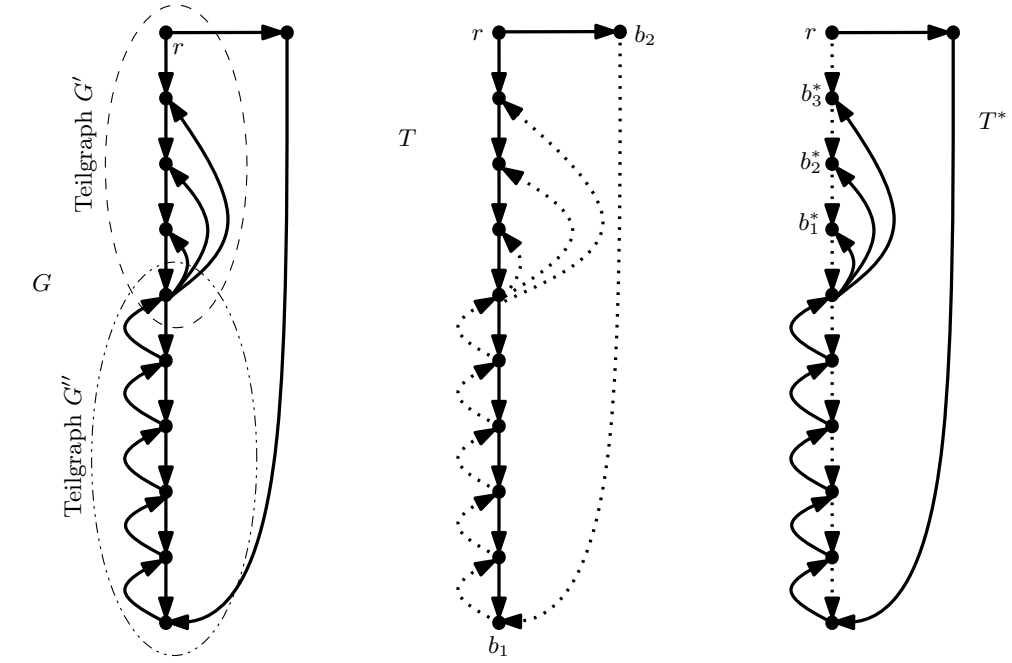
3.2. Übertragung der lokalen Suche auf gerichtete Graphen

Es soll nun untersucht werden, ob sich der in Abschnitt 3.1 angeführte k-Change-Ansatz auch zur Lösung des RMLO-Problems in gerichteten Graphen eignet. Offenbar kann Algorithmus 1 nahezu unverändert für den gerichteten Fall übernommen werden. Dabei wird anstelle eines ungerichteten ein gerichteter Graph und anstelle eines Spannbaums ein Outbranching verwendet. Insbesondere gilt die Analogie für die Definition des k-Change. Die k zu tauschenden Kanten werden nun so gewählt, dass die Eigenschaften eines Outbranchings auch nach dem Tausch erhalten bleiben.

Die Korrektheit des Algorithmus ist offensichtlich, denn da stets aus einem Outbranching ein neues Outbranching konstruiert wird, erfüllt der zurückgegebene Graph die Eigenschaften eines Outbranchings.

Drescher und Vetta [DV10] haben bereits gezeigt, dass der k-Change-Ansatz auf gerichteten Graphen beliebig schlechte Lösungen liefert. Ein negatives Beispiel nach Drescher und Vetta für $k = 7$ ist in Abbildung 3.1 dargestellt. Der obere Teil zeigt die Eingabeinstanz, ein lokales Optimum T für $k = 7$ und ein globales Optimum T^*. Das lokale Optimum T besitzt gerade die zwei Blätter b_1 und b_2. Das globale Optimum T^* hingegen besitzt die drei Blätter b_1^*, b_2^* und b_3^*.

Die Teilbilder a-d zeigen im Einzelnen, dass mindestens ein 8-Change benötigt wird um die Anzahl der Blätter zu erhöhen. Die eingeklammerten Zahlen geben die Abarbeitungsreihenfolge an. Um vom lokalen Optimum T zu einem Outbranching mit mehr Blättern zu kommen, müssen die Kanten in der angegebenen Reihenfolge getauscht werden. Andernfalls wird der Zusammenhang der Initiallösung T zerstört. Im Schritt (1) wird das Blatt b_2 zerstört und Blatt b_2' erstellt. Insgesamt bleibt bei diesem Tausch die Anzahl der Blätter konstant, weshalb er nicht durchgeführt wird (vergleiche Abbildung 3.1a). Ebenso bleibt die Anzahl der Blätter in den Schritten (2) bis (6) konstant (vergleiche Abbildung 3.1b). Abbildung 3.1c zeigt, dass auch Schritt (7) mit Erstellung von Blatt b_2^* und Zerstörung von Blatt b_2'' keine Verbesserung bewirkt. Erst im Schritt (8) bewirkt der Tausch die Erstellung des Blattes b_3^* ohne ein vorher vorhandenes Blatt zu zerstören. Damit verbessert sich die Anzahl der Blätter (vergleiche Abbildung 3.1d). Schritt (9) wird nicht mehr



Eingabeinstanz G und Teilgraphen G', G''

lokales Optimum T für $k = 7$

Optimum T^* in G

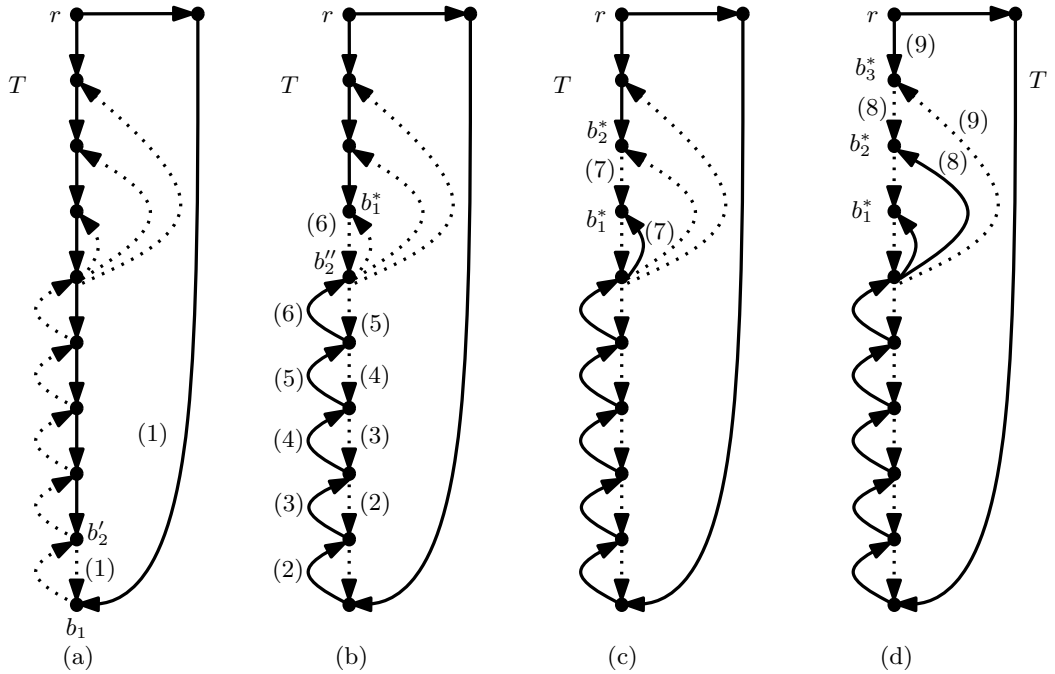


Abb. 3.1.: Eingabe G, ein Spannbaum T, Optimum T^* sowie in a-d Einzelschritte um Initiallösung T zu verbessern.

durchgeführt, da dieser Tausch keine Verbesserung bewirkt (vergleiche ebenfalls Abbildung 3.1d). Damit ist T lokal optimal für $1 \leq k \leq 7$. Verbesserungen treten erst ab einem 8-Change auf.

Offensichtlich kann die Anzahl der Knoten im Teilgraph G'' (vergleiche Abbildung 3.1 oben) beliebig vergrößert werden. Damit kann der Parameter k beliebig hoch gewählt werden.

Ebenso kann die Anzahl der Knoten im Teilgraph G' beliebig erhöht werden. Dabei bleibt die Anzahl der Blätter des lokalen Optimums konstant, das heißt, die Lösung wird beliebig schlecht.

Damit ist ein Algorithmus, der die k-Change-Methode verwendet, nicht zur Lösung des gerichteten RMLO-Problems geeignet.

3.3. Greedy-Verfahren auf ungerichteten Graphen

Der Algorithmus von Solis-Oba [Sol98] ermittelt auf einem zusammenhängenden und ungerichteten Graphen $G = (V, E)$ über ein Greedy-Verfahren einen Spannbaum mit vielen Blättern. Als Ausgangspunkt wird aus einem Knoten $u \in V$ mit $\deg(u) \geq 3$ sowie seiner Nachbarschaft $N(u)$ ein Baum $T' = (V', E')$ erstellt. Dieser wird durch die Expansionsregeln R1, R2 und R3 erweitert (siehe unten sowie Abbildung 3.2). Dabei besitzen R1 und R2 eine höhere Priorität als R3. R1 und R2 sind gleichberechtigt. Die Idee aller Regeln ist sukzessiv Blätter zu zerstören, aber gleichzeitig für ein zerstörtes Blatt mindestens zwei neue Blätter zu generieren.

R1 ist in Abbildung 3.2 auf der linken Seite skizziert. Sie zeigt einen Knoten b, welcher ein Blatt des aktuellen Baumes T' ist. Außerdem besitze der Knoten b im Graphen G mindestens zwei Nachbarn, welche selbst noch nicht zu T' hinzugefügt wurden. Dann werden diese Nachbarn sowie die Kanten, welche die Nachbarn mit b verbinden, zu T' hinzugefügt, in Zeichen $V' \leftarrow V' \cup N(b)$, $E' \leftarrow E' \cup K$ mit $K = \{(b, v) \mid v \in N(b) \setminus V'\}$.

R2 ist in Abbildung 3.2 auf der rechten Seite skizziert. Sie zeigt einen Knoten b, welcher ein Blatt des aktuellen Baums T' ist. Dieser besitze genau einen Nachbarn u, welcher noch nicht in T' enthalten ist. Der Knoten u wiederum besitze in seiner Nachbarschaft $N(u)$ mindestens drei Knoten, welche ebenfalls noch nicht zu T' hinzugefügt wurden. Dann werden der Knoten u, die beschriebenen Nachbarn von u sowie die benötigten Kanten K zu T' hinzugefügt. Benötigt wird die Kante, die den Knoten u mit dem Knoten b verbindet, und die Kanten, die die noch nicht eingefügten Nachbarn von u mit u verbinden. Formal also $V' \leftarrow V' \cup \{u\} \cup N(u)$, $E' \leftarrow E' \cup K$ mit $K = \{(b, u)\} \cup \{(u, v) \mid v \in N(u) \setminus V'\}$.

R3 ist analog zur Regel R2 definiert. Der Unterschied liegt in der Anzahl der Nachbarn von u, welche in Regel R2 mindestens drei betragen, während Regel R3 genau zwei Nachbarn fordert.

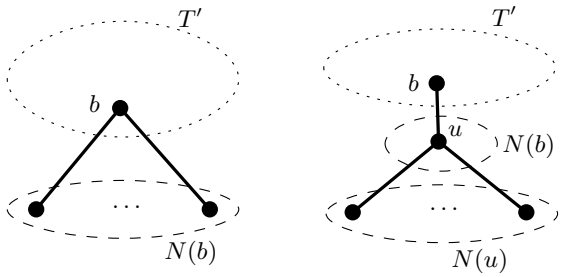


Abb. 3.2.: Expansionsregeln R1 (links) und R2/R3 (rechts).

Diese drei Regeln werden solange angewendet, bis keine Expansion des aktuellen Teilbaums $T' = (V', E')$ mehr möglich ist. Der Baum T' ist zwar Teilbaum von G, jedoch nicht notwendigerweise aufspannend. Der Baum T' wird nun einem zunächst leeren Wald F hinzugefügt. Außerdem werden alle Knoten der Menge V' und Kanten der Menge E' und damit auch alle Kanten, die zu einem Knoten aus V' inzident sind, aus G gelöscht. Schließlich wird im verbleibenden Graphen ein neuer Baum nach obiger Vorgehensweise erstellt und dem Wald F hinzugefügt. Wenn in G kein Knoten u mit Grad $\deg(u) \geq 3$ mehr existiert, werden die in G verbleibenden Knoten dem Wald F als isolierte Knoten hinzugefügt. Dann werden die in F befindlichen Teilbäume und isolierten Knoten iterativ zu einem Spannbaum T zusammengefügt. Dazu werden in jeder Iteration zwei geeignete Bäume gewählt und durch eine Kante aus der Eingabeinstanz G verbunden. Dies wird wiederholt bis ein Spannbaum entstanden ist.

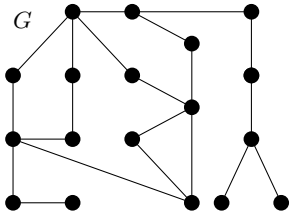
Der Greedy-Algorithmus ist auch in Algorithmus 2 dargestellt. Abbildung 3.3 zeigt seine Anwendung an einem Beispiel. Hier wird deutlich, dass der Algorithmus noch Freiheiten bezüglich der Wahl des ersten Knoten eines Baumes und des zu expandierenden Blattes lässt. Außerdem zeigt das Beispiel, dass ein mit dem Expansionsalgorithmus erstellter Spannbaum nicht zwangsläufig optimal ist.

3.4. Übertragung des Greedy-Verfahren auf gerichteten Graphen

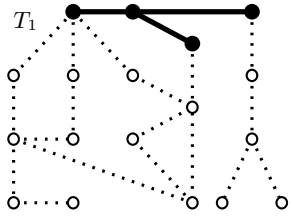
In diesem Abschnitt soll geprüft werden, ob der in Abschnitt 3.3 angeführte Algorithmus 2 auch für gerichtete Graphen verwendet werden kann. Dazu wird Algorithmus 2 in naheliegender Weise auf den gerichteten Fall übertragen. Für die Expansionsregeln (siehe Seite 20) werden statt der Nachbarn die Nachfolger gezählt. Die Expansionsregeln sind in Abbildung 3.4 dargestellt. Die Prioritäten der Expansionsregeln seien wie im ungerichteten Fall gewählt. Die Erstellung eines Baumes startet mit einem Out-2*-Knoten.

Die Übertragung des Greedy-Verfahrens ist unter Algorithmus 3 dargestellt.

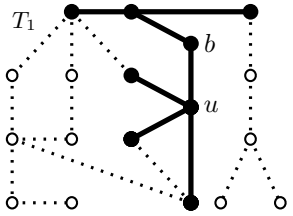
Offensichtlich wirft die Erstellung des Waldes kaum größere Probleme auf. Im Gegensatz dazu steht der Zusammenschluss der einzelnen Bäume zu einem Out-



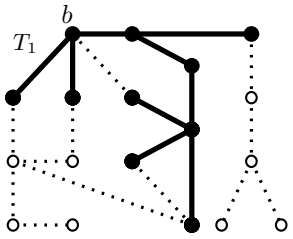
Eingabe G



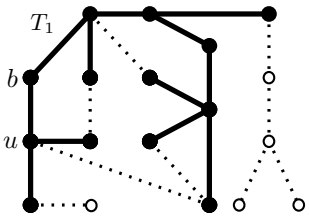
T_1 besteht zunächst aus einem Knoten mit Grad ≥ 3 sowie dessen Nachbarn



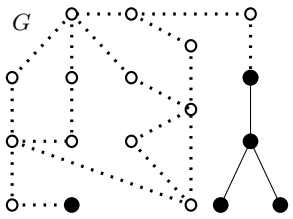
Expansion mit R2



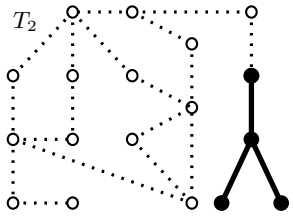
Expansion mit R1



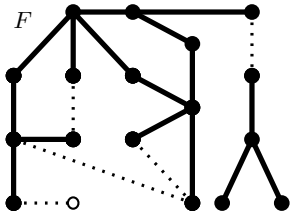
Expansion mit R3, keine weitere Expansion möglich $\Rightarrow$ T_1 wird zum Wald F hinzugefügt



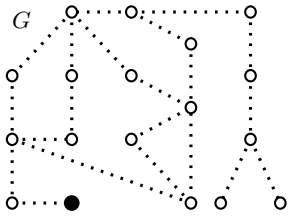
G nach der Löschung von T_1



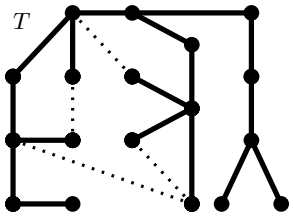
T_2 besteht aus einem Knoten mit Grad ≥ 3 sowie dessen Nachbarn, keine weitere Expansion möglich



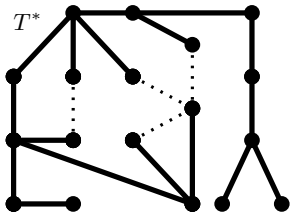
auch T_2 wird zum Wald F hinzugefügt



G ist bis auf einen Knoten leer; dieser wird F hinzugefügt



weitere Kanten führen zu Spannbaum T mit 8 Blättern



anderer Spannbaum T^* mit 9 Blättern

Abb. 3.3.: Eine beispielhafte Anwendung des Algorithmus 2.

Algorithmus 2: MLST-Expansionsalgorithmus(G)

(1) **Input** : zusammenhängender, ungerichteter Graph G
(2) **Output** : Spannbaum T mit vielen Blättern
(3) Wald $F = (V_F, E_F)$, $V_F \leftarrow \emptyset$
(4) **while** *in G existiert ein Knoten v mit Grad* $\deg(v) \geq 3$ **do**
(5) wähle v, die Menge seiner Nachbarn $N(v)$ sowie die benötigten Kanten als T'
(6) **while** *es existiert ein Blatt in T', das expandiert werden kann* **do**
(7) wende von den Expansionsregeln R1, R2 und R3 diejenige mit höchster Priorität an
(8) **end while**
(9) $F \leftarrow F \cup T'$
(10) lösche alle in T' verwendeten Knoten und alle zu diesen Knoten inzidenten Kanten aus G
(11) **end while**
(12) füge die Bäume aus F und die in G verbliebenen Knoten mit Grad < 3 zu einem Spannbaum T zusammen
(13) **return** T

Algorithmus 3: RMLO-Expansionsalgorithmus(G)

(1) **Input** : Wurzelgraph G
(2) **Output** : Outbranching T mit vielen Blättern
(3) $G' \leftarrow G$
(4) Wald $F = (V_F, E_F)$, $V_F \leftarrow \emptyset$
(5) **while** *in G' existiert ein Out-2*-Knoten v* **do**
(6) wähle v, die Menge seiner Nachfolger $N^+(v)$ und die nötigen Kanten als T'
(7) **while** *in T' existiert ein Blatt, das expandiert werden kann* **do**
(8) wende von den Expansionsregeln R1', R2' und R3' diejenige mit höchster Priorität an
(9) **end while**
(10) $F \leftarrow F \cup T'$
(11) lösche alle in T' verwendete Knoten und alle zu diesen Knoten inzidenten Kanten aus G'
(12) **end while**
(13) füge alle in G' verbleibenden Knoten dem Wald F hinzu.
(14) füge die Komponenten des Waldes F mit Prozedur 4 zu einem Outbranching T zusammen
(15) **return** T

Prozedur 4 : erstelleOutbranching(G, F)

(1) **while** $|F| > 1$ **do**
(2) wähle Bäume $T', T'' \in F$, sodass $v' \in V(T'), v'' \in V(T'')$ und $(e = (v', v'') \in E(G)$ oder $e = (v'', v') \in E(G))$
(3) $T \leftarrow T' \cup T'' \cup \{e\}$
(4) $F \leftarrow F \setminus (T' \cup T'')$
(5) $F \leftarrow F \cup T$
(6) **end while**
(7) **return** F

branching. Abbildung 3.5 zeigt, dass dies nicht ohne weitere Modifikationen des Algorithmus möglich ist. Im Beispiel wird zunächst der Knoten v mit $\deg^+(v) = 2$ gewählt, expandiert und dem Wald F hinzugefügt. Anschließend wird mit Wurzel r analog verfahren. Wird für die Verbindung der beiden Teilgraphen eine beliebige Kante verwendet, so kann es passieren, dass Knoten v von der Wurzel r aus nicht erreichbar ist. Das ist der Fall, wenn Kante (u, w) statt (u, v) für den Zusammenschluss gewählt wird.

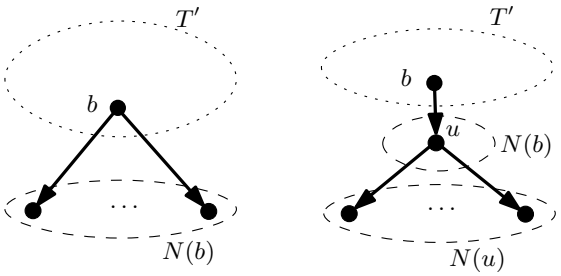


Abb. 3.4.: Expansionsregeln R1' (links) und R2'/R3' (rechts).

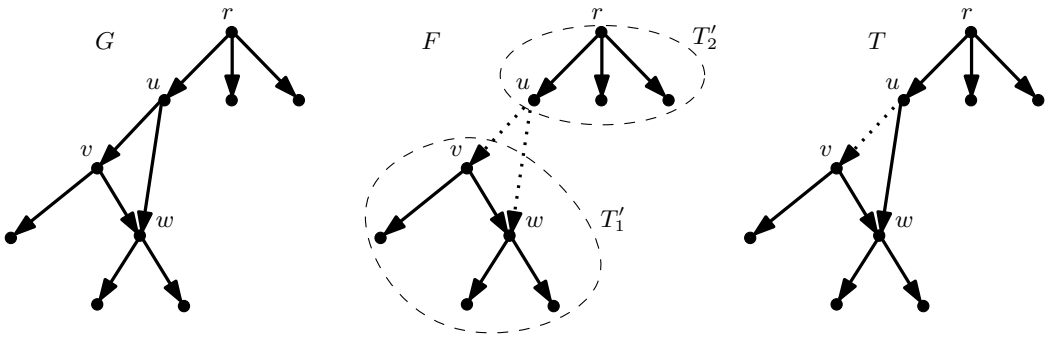


Abb. 3.5.: Eingabeinstanz G, Wald F. In Ausgabe T ist v von r nicht erreichbar.

Auch eine zusätzliche Bedingung, die vorschreibt, dass der erste zu expandierende Knoten immer die Wurzel ist, löst das Problem nicht. Abbildung 3.6 zeigt so

einen Fall, welcher lediglich eine Erweiterung von Abbildung 3.5 ist. Die Expansionsreihenfolge in diesem Beispiel ist r, v, t.

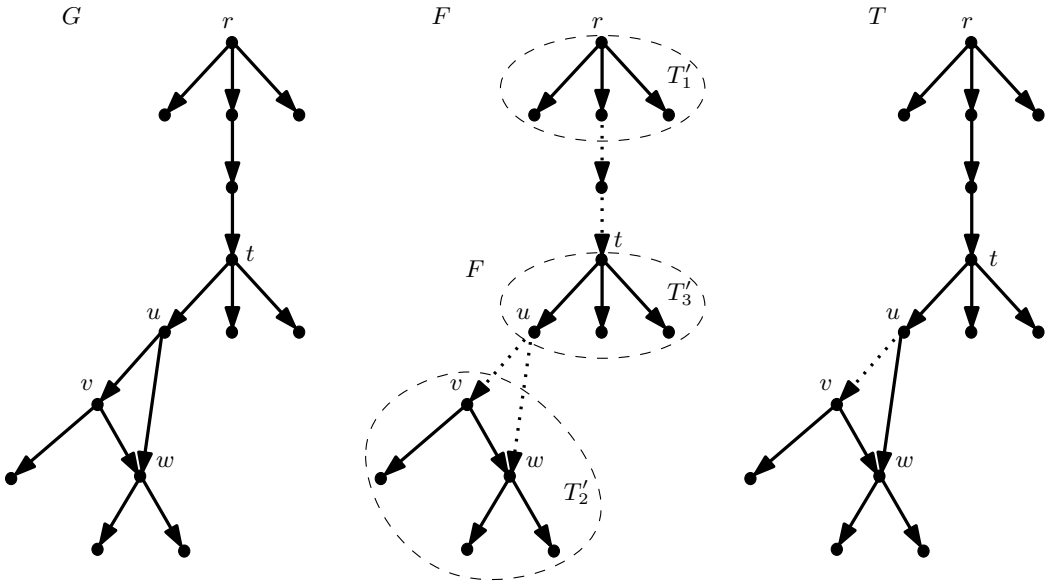


Abb. 3.6.: Eingabeinstanz G, Wald F. In Ausgabe T ist v von r nicht erreichbar, wenn Wurzel r zuerst expandiert wird.

Zunächst scheint das Problem des fehlenden Zusammenhangs durch das Hinzufügen einer weiteren Bedingung – wie sie Prozedur 5 verwendet – behoben zu sein. In dieser wird vorgeschrieben, dass der Zusammenschluss der Bäume aus F nur mit Kanten durchgeführt wird, die in einem Baum des Waldes F starten und in der Wurzel eines anderen Baumes des Waldes F enden.

Prozedur 5 : erstelleOutbranching*(G, F)

(1) **while** $|F| > 1$ **do**
(2) verbinde zwei Bäume T' und T'' mittels der Kante $e = (v_1, v_2) \in E(G)$ zu einem Baum T, wenn v_2 die Wurzel von T' oder T'' ist
(3) $F \leftarrow F \setminus (T' \cup T'')$
(4) $F \leftarrow F \cup T$
(5) **end while**
(6) **return** F

Abbildung 3.7 zeigt die Anwendung von Algorithmus 3 unter Verwendung von Prozedur 5. Als Eingabeinstanz ist hier der Graph G gewählt. Die Wurzel r besitzt Ausgangsgrad $\deg^+(r) = 3$. Sie und ihre Kinder bilden T_1'. Der Knoten w kann nicht expandiert werden. Der Baum T_1' wird dem Wald F hinzugefügt. Mit Knoten r_2 mit $\deg^+(r) = 4$ wird analog verfahren. Schließlich werden die Knoten z_1 und z_2 dem Wald F als isolierte Knoten hinzugefügt.

Nun wird Prozedur 5 angewendet, wobei für die Regel "von Baum in Wurzel" die Wurzeln z_1, z_2 und r_2 zur Verfügung stehen. Offensichtlich muss z_2 über die Kante (w, z_2) und z_1 über die Kante (v, z_1) angeschlossen werden, denn $\deg_G^-(z_1) = \deg_G^-(z_2) = 1$. Für die verbleibende Wurzel r_2 gibt es keine sinnvolle Anbindungsmöglichkeit, denn die einzige Kante, die in die Wurzel mündet, stellt keinen Zusammenhang her und verursacht zusätzlich die Entstehung eines Kreises. Prozedur 5 würde nicht terminieren. Es wäre $|F| > 2$, wobei keine Kante existiert, die den in Prozedur 5 gestellten Bedingungen genügt.

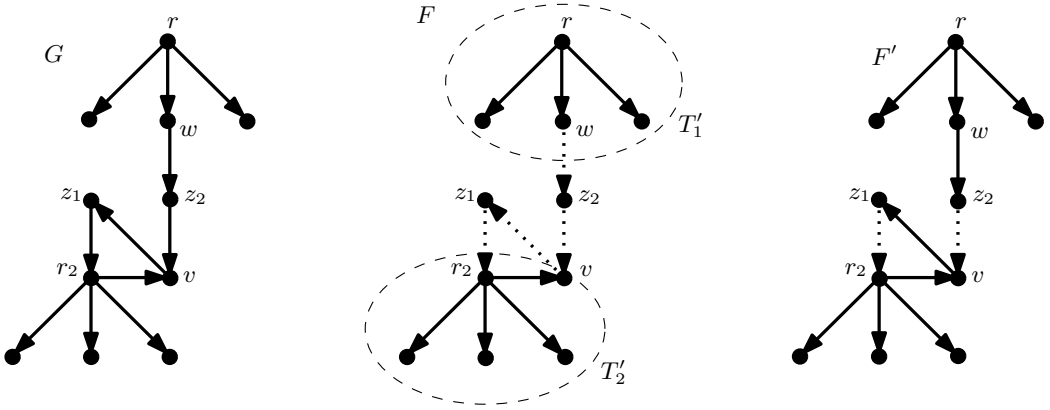


Abb. 3.7.: Eingabegraph G, Wald F und teilweise durchgeführter Zusammenschluss der Bäume in F'.

Damit zeigt Abbildung 3.7, dass der Expansionsalgorithmus nicht ohne Weiteres auf gerichtete Graphen übertragbar ist. In Abschnitt 4.4 wird seine Anwendbarkeit auf gerichtete azyklische Graphen untersucht.

3.5. Zwischenfazit

Abschnitt 3.2 und 3.4 haben gezeigt, dass die intuitive Übertragungen von Algorithmen, die das ungerichtete RMLO-Problem lösen, auf Algorithmen, die das gerichtete RMLO-Problem lösen, nicht ohne Weiteres möglich ist. Im weiteren Verlauf der Arbeit wird das gerichtete RMLO-Problem nicht weiter behandelt. Die Frage nach einem Algorithmus zur Lösung des gerichteten RMLO-Problems bleibt in dieser Arbeit also offen. Dennoch ist ein prinzipielles Problem bei der Lösung des gerichteten RMLO-Problems erkennbar. Dieses wird in Abbildung 3.5 (siehe Abschnitt 3.4) deutlich. Die Schwierigkeit des Problems liegt in der Behandlung von Kreisen. Das Beispiel in Abbildung 3.5 gibt einen Hinweis darauf, dass bestimmte Knoten auf Kreisen gesondert behandelt werden sollten. Dabei handelt es sich um die *Gelenkpunkte*.

Eine naheliegende Idee ist daher sicher zu stellen, dass ein Gelenkpunkt, der auf einem Kreis liegt, in der korrekten Richtung durchlaufen wird. Für das Beispiel in Abbildung 3.5 wäre so die Durchlaufreihenfolge der problematischen Kno-

ten z_2, v, z_1, r_2 für das Outbranching vorgeben. Damit wäre der mit dem bisherigen Algorithmus nicht vorhandene Zusammenhang des Outbranchings gewährleistet.

4. Lösungsansätze für azyklische Graphen

In Kapitel 3 wurde gezeigt, dass die Lösung des RMLO-Problems mit den bisher bekannten Methoden für ungerichtete Graphen (übertragen auf den gerichteten Fall) nicht ohne Weiteres möglich ist. Daher werden im Folgenden Graphen einer speziellen Graphklasse untersucht, nämlich gerichtete azyklische Graphen.

Es könnte die Vermutung aufkommen Azyklizität von Graphen sei eine starke Einschränkung, die das Problem entscheidend vereinfachen müsste. Dass dies im Allgemeinen nicht der Fall ist, lehrt das bekannte Problem Steiner Tree (kurz ST-Problem). Das ST-Problem gehört wie das RMLO-Problem auch zur Klasse der Spannbaumprobleme.

Für den ungerichteten Fall des ST-Problems sind gute Approximationsalgorithmen bekannt. Einer der bekanntesten Approximationsalgorithmen besitzt Faktor $\alpha < 1,55$ [RZ00]. Die Lösung des gerichteten ST-Problems ist wesentlich schwerer. Der bislang beste Approximationsalgorithmus besitzt eine Güte von $\mathcal{O}(n^\epsilon)$, wobei $\epsilon > 0$ und n die Anzahl der Knoten der Eingabeinstanz ist [CCC+99]. Zur Lösung des ST-Problems in azyklischen gerichteten Graphen gibt es zwar einen spezialisierten Approximationsalgorithmus, dieser besitzt jedoch ebenfalls nur eine Güte von $\mathcal{O}(n^\epsilon)$ [Zel97]. Es lässt sich sogar zeigen, dass sich das azyklische gerichtete ST-Problem nicht besser als mit Faktor $\Omega(\log n)$ approximieren lässt [Zel97].

Nach derzeitigem Wissensstand lässt sich das ST-Problem auf azyklischen gerichteten Graphen also nicht besser als auf allgemeinen gerichteten Graphen lösen. Zweitens ist das azyklische gerichtete ST-Problem wesentlich schwerer als das ungerichtete ST-Problem.

Im weiteren Verlauf der Arbeit wird sich herausstellen, dass sich das RMLO-Problem und das ST-Problem in diesem Punkt unterscheiden. Die Azyklizität erlaubt eine signifikante Verbesserung gegenüber allgemeinen gerichteten Graphen. In dieser Arbeit wird ein 2-Approximationsalgorithmus entwickelt. Dieser Faktor entspricht dem besten bekannten Approximationsfaktor für das ungerichtete RMLO-Problem von Solis-Oba [Sol98]. Der hier entwickelte Algorithmus und die verwendeten Beweistechniken sind sogar einfacher als diejenigen von Solis-Oba. Im Gegensatz zum ST-Problem scheint der azyklische gerichtete Fall des RMLO-Problems leichter zu sein als der ungerichtete Fall.

Außerdem bleibt das RMLO-Problem trotz der Kreisfreiheit NP-schwer [AFG+09]. Im Folgenden wird gezeigt, dass für dieses Problem auch kein PTAS existiert.

4.1. MaxSNP-Schwere des MLST-Problems

Es gibt verschiedene Klassen für Probleme, welche nicht beliebig gut approximierbar sind. Eine dieser Klassen ist die von Papadimitriou und Yannakakis [PY91] vorgestellte Klasse MaxSNP. Galbiati, Maffioli und Morzenti [GMM94] haben gezeigt, dass das Problem Maximum Leaf Spanning Tree (ungerichteter Fall des RMLO-Problems, kurz auch MLST-Problem) MaxSNP-schwer ist. Um die MaxSNP-Schwere des MLST-Problems zu beweisen haben Galbiati et al. die so genannte *L-Reduktion* verwendet. Im Folgenden wird zunächst die L-Reduktion definiert. Danach wird gezeigt wie Galbiati et al. diese verwendet haben um die MaxSNP-Schwere des MLST-Problems zu beweisen.

Definition 4 (L-Reduktion)**.** Seien P und Q zwei Optimierungsprobleme. Eine *L-Reduktion* von P auf Q ist durch die zwei Abbildungen f und g gegeben. Sei I eine Instanz des Problems P. Diese wird mittels f auf eine Instanz I' des Problems Q abgebildet, also $f : P \to Q$. Die Abbildung wird auch als *Transformation* bezeichnet. Aus jeder gültigen Lösung L' des Problems Q für die Instanz I' lässt sich durch die Abbildung $g : L' \mapsto L$ ("Rücktransformation") eine Lösung L für die Instanz I des Problems P ermitteln. Die beiden Abbildungen müssen in Polynomialzeit berechenbar sein. Außerdem muss es zwei konstante Faktoren $\alpha, \beta > 0$ geben, sodass die Bedingungen 4.1 und 4.2 erfüllt sind.

Es gibt einen konstanten Wert $\alpha > 0$, sodass für alle Instanzen I

$$OPT_Q(I') \leq \alpha OPT_P(I). \tag{4.1}$$

Es gibt einen konstanten Wert $\beta > 0$, sodass für alle zulässigen Lösungen L' der Instanz I'

$$|OPT_P(I) - L| \leq \beta |OPT_Q(I') - L'|. \tag{4.2}$$

Ausschlaggebend sind hier also Unterschiede zwischen optimalen und allgemeinen Lösungen.

Sind diese Bedingungen erfüllt und ist für das Problem Q ein PTAS bekannt, dann existiert auch für das Problem P ein PTAS.

Galbiati et al. verwenden als Problem P das Problem in einem ungerichteten, gradbeschränkten Graphen $G = (V, E)$ eine kleinste dominierende Menge $D \subseteq V$ zu bestimmen. Für das Problem Q verwenden sie das MLST-Problem.

Für die Transformation sei $G = (V, E)$ als Eingabegraph mit $V = \{v_1, v_2, \ldots, v_n\}$ und $|V| = n$ gewählt. Sei $G' = (V', E')$ der transformierte Graph. Um die Menge der Knoten V' zu erhalten verdreifache jeden Knoten aus V und füge außerdem die beiden Knoten α_0 und α_1 hinzu, das heißt, $V' = \{\alpha_0, \alpha_1, 1_1, 1_2, \ldots, 1_n, 2_1, \ldots, 2_n, 3_1, \ldots, 3_n\}$. Dabei deuten die Indizes i an aus welchem Knoten $v_i \in V$ der transformierte Knoten entstanden ist.

Für die Menge der Kanten E' wird zunächst ein Gerüst erstellt. Es ist in Abbildung 4.1 dargestellt. Um dieses Gerüst zu erhalten verbinde die drei trans-

formierten Knoten $1_i, 2_i, 3_i$ jedes Knoten $v_i \in V$ sowie α_0 über die Kanten $(\alpha_0, 1_i), (1_i, 2_i), (2_i, 3_i)$. Füge die Kante (α_1, α_0) hinzu. Außerdem sei die Zeile, in der die erste Kopie 1_i jedes Knoten $v_i \in V$ steht, mit *Ebene 1* bezeichnet (vergleiche Abbildung 4.1).

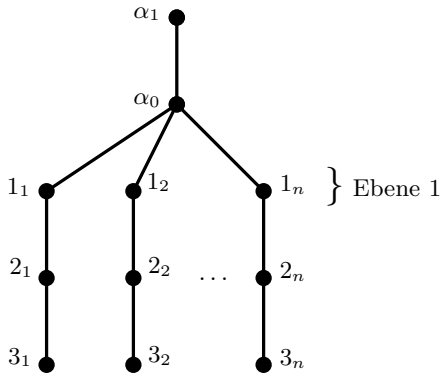


Abb. 4.1.: Schematisches Gerüst des transformierten Graphen G'.

Abbildung 4.2 zeigt an einem Beispiel den letzten Schritt der Transformation. In diesem letzten Schritt werden die Kanten des Eingabegraphen G in das Gerüst des transformierten Graphen G' eingefügt. Erstelle dazu für jede Kante $e = (v_k, v_i) \in E$ die zwei Kanten $(1_k, 2_i)$ und $(1_i, 2_k)$.

Die Konstruktion des Gerüsts und die Übertragung der Kanten entsprechen der Abbildung f der L-Reduktion.

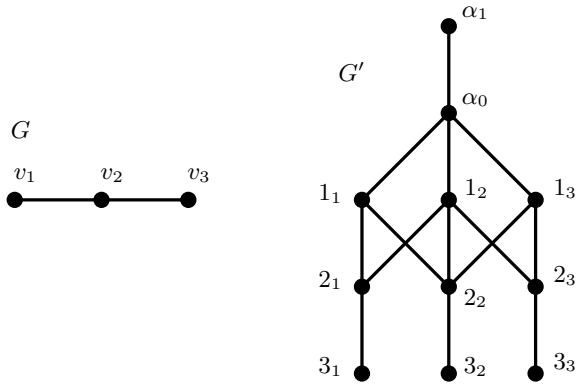


Abb. 4.2.: Vom Eingabegraphen G zum transformierten Graphen G' – Abbildung f der L-Reduktion.

Zur Beschreibung der Abbildung g der L-Reduktion, betrachte im transformierten Graphen G' einen beliebigen Spannbaum T. Wähle alle inneren Knoten der Ebene 1 des Spannbaums T als Menge D'. Diese wird "rücktransformiert", das heißt, setze $D := \{v_d \mid 1_d \in D'\}$. Schließlich wird die Menge D als dominierende Menge für G gewählt.

Um leichter zeigen zu können, dass die gewählte Menge D tatsächlich eine dominierende Menge in G ist, werden im Spannbaum T zunächst einige Kanten getauscht. Für jeden Knoten $v_i \in V$ wird geprüft, ob die Kante $(\alpha_0, 1_i)$ im Spannbaum T enthalten ist. Falls nicht, wird die Kante $(\alpha_0, 1_i)$ zu T hinzugefügt. Im Gegenzug wird die angrenzende Kante $(1_i, 2_l)$ mit $1 \leq l \leq n$ gelöscht. Dadurch bleibt der Spannbaum kreisfrei. Galbiati et al. zeigen außerdem, dass sich die Blätter von T durch die Vertauschungen der Kanten nicht ändern. Sei T' der neue, zu T äquivalente Spannbaum.

Falls der Spannbaum T' ein Blatt 1_k mit $1 \leq k \leq n$ besitzt, dann muss der Knoten 2_k von einem Knoten 1_j mit $1 \leq j \leq n$ und $k \neq j$ erreicht werden. Der Knoten 1_j ist also innerer Knoten der Ebene 1 des Spannbaums T'. Damit gilt $1_j \in D'$ beziehungsweise $v_j \in D$. Aus der Konstruktion des Graphen G' folgt, dass die Knoten v_j und v_k in G benachbart sind, denn 1_j und 2_k sind benachbart in G'. Somit dominiert der Knoten v_j den Knoten v_k in G. Da der Knoten 2_i jedes Knoten $v_i \in V$ aufgrund der Spannbaumeigenschaft von einem Knoten aus Ebene 1 in T' erreicht wird, werden alle Knoten aus V dominiert.

Zuletzt zeigen Galbiati et al. die Existenz der beiden Konstanten α und β, die den Eigenschaften 4.1 und 4.2 genügen.

4.2. Das azyklische RMLO-Problem ist MaxSNP-schwer

Galbiati et al. [GMM94] haben gezeigt, dass das MLST-Problem MaxSNP-schwer ist. Damit existiert für das MLST-Problem kein PTAS. Für den Beweis wurde eine so genannte L-Reduktion verwendet. Das grundsätzliche Vorgehen bei einer L-Reduktion wurde in Abschnitt 4.1 vorgestellt. Dort sind insbesondere auch die Transformationen dargestellt, die für den Beweis der MaxSNP-Schwere des MLST-Problems verwendet wurden. Nun soll gezeigt werden, dass das RMLO-Problem in azyklischen Graphen auch MaxSNP-schwer ist. Dazu kann eine L-Reduktion mit nahezu denselben Transformationen wie beim MLST-Problem verwendet werden. Infolgedessen brauchen die von Galbiati et al. durchgeführten Rechnungen zum Beweis der MaxSNP-Schwere des MLST-Problems nicht angepasst werden.

Als Ausgangsproblem P wird wieder die Suche nach einer kleinsten dominierende Menge in einem ungerichteten, gradbeschränkten Graphen verwendet.

Die Transformation des Eingabegraphen $G = (V, E)$ kann nun zunächst wie beim Beweis der MaxSNP-Schwere des MLST-Problems durchgeführt werden (vergleiche Seite 30). Ebene 1 enthalte wie in Abschnitt 4.1 die erste Kopie 1_i jedes Knoten $v_i \in V$ (vergleiche Abbildung 4.1). Analog seien die Kopien 2_i und 3_i mit Ebene 2 beziehungsweise Ebene 3 bezeichnet. Ebene 0 enthalte den Knoten α_0. Ebene -1 enthalte den Knoten α_1.

Die aus Abschnitt 4.1 bekannte Transformation der Eingabeinstanz G wird beim Beweis der MaxSNP-Schwere des RMLO-Problems um einen Schritt erweitert. Die Kanten des transformierten Graphen G' werden so gerichtet, dass sie stets von Ebene j in Ebene $j + 1$ für $j = -1, 0, 1, 2$ zeigen. Damit ist der transformierte

Graph offensichtlich azyklisch. Außerdem wird der Knoten α_1 als Wurzel gewählt.

Erstelle nun ein beliebiges Outbranching T im transformierten Graphen G'. Offensichtlich werden die Knoten einer Ebene $j+1$ per Konstruktion nur von ihrer vorhergehenden Ebene j für $j = -1, 0, 1, 2$ erreicht. Ebene -1 und Ebene 0 enthalten jeweils nur einen Knoten, nämlich den Knoten α_1 beziehungsweise den Knoten α_0. Damit kann weder α_1 noch α_0 ein Blatt in T sein. Ebenso kann in Ebene 2 kein Blatt liegen. Andernfalls wäre ein Knoten aus Ebene 3 isoliert. Demnach können die Blätter des Outbranchings T nur in Ebene 1 und Ebene 3 liegen.

Das Outbranching T ist also äquivalent zu dem Spannbaum T', welcher beim Beweis der MaxSNP-Schwere des MLST-Problems durch die Vertauschung der Kanten entsteht. Damit ist die Argumentation, dass mit den inneren Knoten von Ebene 1 des Spannbaums T' eine dominierende Menge für die Eingabeinstanz G ermittelt werden kann (vergleiche Seite 31), auch für das Outbranching T korrekt.

Bei einer L-Reduktion muss stets die Existenz zweier Konstanten α und β nachgewiesen werden. Beim Beweis der MaxSNP-Schwere des MLST-Problems flossen beim Existenzbeweis von α und β insbesondere zwei Werte mit ein. Der erste Wert ist die Kardinalität einer kleinsten dominierenden Menge (im gradbeschränkten Graphen) des Eingabegraphen G. Dieser Wert ist offensichtlich nur von der Eingabeinstanz G abhängig. Damit kann im Beweis der MaxSNP-Schwere des RMLO-Problems derselbe Wert wie im Beweis von Galbiati et al. verwendet werden.

Der zweite Wert ist die maximale Blattanzahl im Spannbaum des transformierten Graphen G'. Ein RMLO von G' besitzt gerade ein Blatt weniger als ein äquivalenter MLST von G', nämlich α_1. Diese konstante Additive fällt bei den Beweisen von Galbiati et al. jedoch nicht weiter ins Gewicht.

Insgesamt bleiben damit die von Galbiati et al. durchgeführten Rechnungen unverändert korrekt.

Durch die Verwendung einer L-Reduktion kann nun darauf geschlossen werden, dass für das RMLO-Problem kein PTAS existiert. Würde für das RMLO-Problem ein PTAS existieren, dann würde auch für das Problem eine kleinste dominierende Menge in einem ungerichteten, gradbeschränkten Graphen zu bestimmen ein PTAS existieren. Von letzterem Problem ist jedoch bewiesen, dass kein PTAS existiert, falls $P \neq NP$ [PY91].

Motiviert von diesem Ergebnis werden im Folgenden zwei Approximationsalgorithmen mit konstantem Approximationsfaktor für das *azyklische* RMLO-Problem entworfen.

4.3. In-2-Algorithmus

Gegeben sei ein azyklischer Graph $G = (V, E)$ mit Wurzel r. Die Menge aller In-2*-Knoten (vergleiche Definition 3, Seite 13) in G sei mit V_2 und die Menge aller In-1-Knoten (siehe ebenfalls Definition 3, Seite 13) in G mit V_1 bezeichnet. Die beiden Mengen V_1 und V_2 sind offensichtlich disjunkt. Es gilt $|V_1| + |V_2| = |V| - 1$, denn die Wurzel besitzt Eingangsgrad $\deg^-(r) = 0$ und damit $r \notin V_1 \cup V_2$.

Satz 4.1 (Daligault und Thomassé [DT09])**.** *Sei G ein azyklischer Wurzelgraph mit $|V_2|$ In-2*-Knoten und einer Wurzel mit $\deg(r) \geq 2$, dann besitzt G ein Outbranching mit wenigstens $(|V_2| + \deg(r) + 2)/3$ Blättern.*

Im Folgenden wird untersucht, ob sich Satz 4.1 zur Lösung des RMLO-Problems in azyklischen Graphen nutzen lässt.

Zunächst ist festzustellen, dass der Beweis zu Satz 4.1 konstruktiv ist. Daher lässt er sich leicht zu einem Algorithmus umformen. Dieser ist aus Gründen der Vollständigkeit in Algorithmus 6 dargestellt. Er verwendet die drei Prozeduren 7, 8 und 9, welche ebenfalls nur der Vollständigkeit halber auf den Seiten 35 und 36 beschrieben sind.

An dieser Stelle sei ausdrücklich darauf hingewiesen, dass sich der Hauptteil von Prozedur 7 und die gesamte Prozedur 8 unmittelbar aus dem Beweis von Daligault und Thomassé ergeben. Ein Verständnis der Vorgehensweise des Algorithmus ist zum Verständnis der Beweise in dieser Arbeit nicht notwendig. Daher wird auf den Algorithmus und auch auf den Beweis von Daligault und Thomassé hier nicht näher eingegangen. Entscheidend ist, dass mit Algorithmus 6 ein Outbranching, welches Satz 4.1 genügt, in Polynomialzeit berechnet werden kann (vergleiche Lemma 4.2).

Soll Algorithmus 6 als Approximationsalgorithmus mit konstanter Güte verwenden werden, liegt das Problem bei der Abschätzung dieses Approximationsfaktors. Die angegebene Schranke von mindestens $(|V_2|+\deg(r)+2)/3$ Blättern hängt insbesondere von der Anzahl $|V_2|$ der In-2*-Knoten ab. Gibt es viele dieser In-2*-Knoten, dann liefert Satz 4.1 eine gute untere Schranke.

Existieren hingegen nur wenig In-2*-Knoten, dann gibt es viele In-1-Knoten, das heißt, Knoten, die genau eine eingehende Kante besitzen. Die besondere Eigenschaft dieser Knoten ist, dass die eingehende Kante bei der Erstellung eines Outbranchings in jedem Fall verwendet wird. Andernfalls wäre der Baum nicht zusammenhängend, da der Vorgänger eines In-1-Knoten eindeutig ist. Gibt es viele Kanten, die sicher im Outbranching enthalten sind, dann gibt es verhältnismäßig wenig Spielräume bei der Wahl der übrigen Kanten. Darum liefert hier intuitiv schon ein beliebiges Outbranching gute Ergebnisse.

Da Algorithmus 6 in jedem Fall ein Outbranching liefert, kann er für beide Fälle verwendet werden.

Lemma 4.1. *Algorithmus 6 liefert ein Outbranching.*

Beweis. Die Zeilen 9 bis 14 von Prozedur 7 und die gesamte Prozedur 8 sind eine Übertragung des konstruktiven Beweises des Satzes 4.1 von Daligault und Thomassé. Daligault und Thomassé konstruieren aus einer Eingabeinstanz ein Outbranching.

Die Zeilen 1 bis 8 von Prozedur 7 sind eine Vorbereitung, welche den Graphen G so modifizieren, dass er den Voraussetzungen von Satz 4.1 genügt. Durch die Modifikation erhält die Wurzel r in jedem Fall Grad $\deg(r) \geq 2$. Der Beweis von Daligault und Thomassé bleibt unverändert richtig, da auf dem Pfad von der Wurzel r nach v

Algorithmus 6: In-2-Algorithmus(G)

(1) **Input** : azyklischer Wurzelgraph G
(2) **Output** : Outbranching T
(3) $(G, V_2'', W, W') \leftarrow$ Prozedur 7 `// Vorverarbeitung`(G)
(4) $T' \leftarrow$ Prozedur 8 `// Hauptverarbeitung`(G, V_2'', W, W')
(5) $T \leftarrow$ Prozedur 9 `// Nachbereitung`(T')
(6) **return** T

Prozedur 7 : Vorverarbeitung(G)

`// allgemeine Vorverarbeitung`
(1) **if** *Graph G ist ein Pfad $P(r, v)$ von Wurzel r zum Knoten $v \in V$* **then**
(2) **return** G
(3) **end if**
(4) **if** $\deg(r) = 1$ **then**
(5) sei $P(r, v)$ mit $\deg^+(v) \geq 2$ der Pfad von r zum ersten Knoten v mit mehr als einem Nachfolger
(6) $r' \leftarrow r$
(7) $r \leftarrow v$
(8) **end if**

`// Vorverarbeitung nach Daligault und Thomassé`
(9) lösche beliebige Kanten aus Graph $G = (V, E)$, sodass G ausschließlich In-1- und In-2-Knoten besitzt
(10) sei V_1 die Menge aller In-1-Knoten aus V
(11) sei V_2 die Menge aller In-2-Knoten aus V
(12) sei W die Menge aller Knoten aus $N^-(v)$ mit $v \in V_1$
`//` W `ist also die Menge aller Vorgänger aller Knoten aus` V_1
(13) sei V_2' die Menge aller Knoten $v' \in N^+(w)$ mit $w \in W$ und $\deg^-(v') = 2$
`//` V_2' `ist also die Menge aller In-2-Knoten, die Nachfolger von einem Knoten aus` W `sind`
(14) sei $V_2'' = V_2 \setminus V_2'$
`//` V_2'' `ist also die Menge aller In-2-Knoten, die nicht Nachfolger von einem Knoten aus` W `sind`
(15) sei S die Menge aller Senken
(16) sei $W' = V \setminus (W \cup S)$
`//` W' `ist also die Menge aller Knoten, die nicht Vorgänger eines In-1-Knoten und keine Senke sind`
(17) **return** (G, V_2'', W, W')

Prozedur 8 : Hauptverarbeitung(G, V_2'', W, W')

```
// Hauptprozess nach Daligault und Thomassé
```
(1) erstelle einen bipartiten Graphen $G_{\text{bip}} = (V_2'' \uplus W', E_{\text{bip}})$ mit $E_{\text{bip}} = \{(w', v') \mid v' \in V_2'',\ w' \in W' \text{ falls } (w', v') \in E\}$

(2) erstelle einen Graphen $G_{\text{dom}} = (V_{\text{dom}}, E_{\text{dom}})$, wobei $V_{\text{dom}} = W'$ und $\{d_1, d_2\} \in E_{\text{dom}}$, wenn d_1 und d_2 einen gemeinsamen Nachbarn in G_{bip} besitzen

(3) $D' \leftarrow \emptyset$

// erstelle Menge D', sodass jeder Knoten aus V_2'' in G mit einem Knoten aus D' benachbart ist

(4) **while** $V_{dom} \neq \emptyset$ **do**

(5) **if** *in* G_{dom} *existiert ein Knoten* v *mit* $\deg(v) \geq 2$ **then**

(6) $D' \leftarrow D' \cup \{v\}$

(7) lösche v und die zu v inzidenten Kanten aus G_{dom}

(8) **else**

(9) wähle einen beliebigen, nicht isolierten Knoten v

(10) $D' \leftarrow D' \cup \{v\}$

(11) lösche v und die zu v inzidenten Kanten aus G_{dom}

(12) **end if**

(13) **end while**

(14) $D \leftarrow D' \cup W$

// D ist zusammenhängende dominierende Menge von G

(15) erstelle $G[D]$

(16) suche einen Spannbaum T' in $G[D]$

(17) füge alle Knoten aus $V \setminus D$ als Blätter zu T' hinzu

(18) **return** T'

Prozedur 9 : Nachbereitung(T')

```
// Nachbereitung
```
(1) **if** *die Vorbereitung wurde durchgeführt* **then**

(2) T ist ein Outbranching, das entsteht, indem T' die in der Vorverarbeitung übersprungen Knoten und Kanten wieder hinzugefügt werden

(3) $r \leftarrow r'$ in T

(4) **end if**

(5) **return** T

keine weiteren In-2*-Knoten liegen. Damit wird kein mögliches Blatt verhindert. Bei der Abschätzung der Blattanzahl im konstruierten Outbranching wird die neue Wurzel r mit $\deg(r) \geq 2$ verwendet.

Zeile 1 von Prozedur 7 stellt einen Sonderfall dar. Er wird ausgeschlossen, damit in Zeile 4 von Prozedur 7 kein Fehlschlag auftritt.

Die Nachbereitung in Zeile 1 von Prozedur 9 fügt die in der vorbereitenden Prozedur 7 übersprungenen Knoten und Kanten wieder hinzu. Alle Knoten der Eingabeinstanz werden erreicht. □

Für den folgenden Beweis sei $|V| = n$ die Anzahl der Knoten und $|E| = m$ die Anzahl der Kanten der Eingabeinstanz $G = (V, E)$.

Lemma 4.2. *Algorithmus 6 benötigt Laufzeit $\mathcal{O}(n + m)$.*

Beweis. Die vorbereitenden Schritte in Zeile 1 und 4 von Prozedur 7, die Vorbereitung von Daligault und Thomassé in den Zeilen 10 bis 14 von Prozedur 7 und die Nachbereitung in Zeile 1 von Prozedur 9 benötigen jeweils lineare Laufzeit.

Die Kantenlöschung in Zeile 9 von Prozedur 7 kann in $\mathcal{O}(m)$ Zeit abgearbeitet werden.

Für Prozedur 8 wird eine Laufzeit von $\mathcal{O}(n + m)$ benötigt. Die Erstellungen des bipartiten Graphen in Zeile 1, des Graphen in Zeile 2, des induzierten Teilgraphen in Zeile 15 inklusive dem Hinzufügen der Blätter in Zeile 17 sowie die while-Schleife ab Zeile 4 benötigen jeweils eine Laufzeit von $\mathcal{O}(n+m)$. Die Erstellung des Spannbaums in Zeile 16 kann mit Tiefensuche durchgeführt werden. Dann beträgt die Laufzeit auch hier $\mathcal{O}(n + m)$.

Damit hat der Algorithmus insgesamt eine Laufzeit von $\mathcal{O}(n + m)$. □

Lemma 4.3. *Algorithmus 6 besitzt eine Güte von 4.*

Beweis. Als Eingabe sei der azyklische Graph $G = (V, E)$ mit Wurzel r gegeben. Sei V_2 die Menge aller In-2*-Knoten in G und V_1 die Menge aller In-1-Knoten in G. Die beiden Mengen sind offensichtlich disjunkt. Es gilt $|V_1| + |V_2| = |V| - 1$, denn $\deg^-(r) = 0$.

Sei L^* die Menge aller Blätter in einem RMLO und L die Menge aller Blätter, die Algorithmus 6 liefert. Sei $\alpha := |V_2|/|L^*|$. Es existieren zwei mit 4.3 und 4.4 bezeichnete Schranken. Sie werden nachfolgend bewiesen.

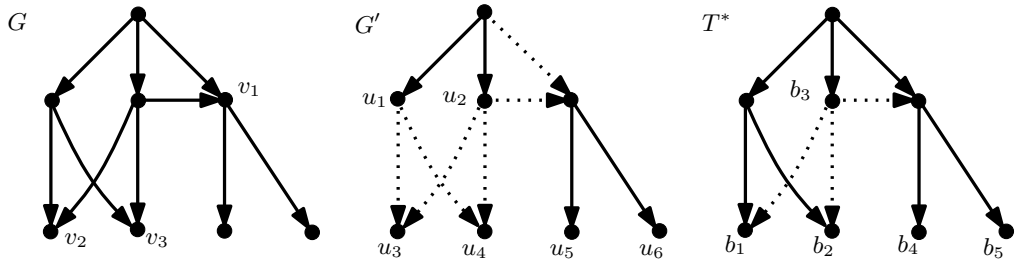
$$|L| \geq \frac{\alpha}{3}|L^*| \tag{4.3}$$

$$|L| \geq |L^*|(1 - \alpha). \tag{4.4}$$

Nachstehende Gleichungskette zeigt, dass die Schranke 4.3 aus Satz 4.1, dem Grad der Wurzel $\deg(r) \geq 2$, den Satz 4.1 fordert, und der Definition von α folgt.

$$|L| \geq \frac{|V_2| + \deg(r) + 2}{3} \geq \frac{|V_2| + 2 + 2}{3} \geq \frac{|V_2|}{3} = \frac{\alpha|L^*|}{3}.$$

Abbildung 4.3 soll helfen die Schritte des nun folgenden Beweises von Schranke 4.4 besser zu verstehen.



Eingabegraph G | Graph G' mit Kanten, die sicher im Outbranching sind | RMLO von G

Abb. 4.3.: Beispiel zum Beweis von Schranke 4.4. Es ist $V_2 = \{v_1, v_2, v_3\}$, $L' = \{u_1, \ldots, u_6\}$ und $L^* = \{b_1, \ldots, b_5\}$.

Zum Beweis von Schranke 4.4 wird ein Graph $G' = (V, E')$ mit $E' = \{(u, v) \in E \mid v \in V_1\}$ verwendet. Dieser Graph ist ein Wald, der nur Kanten besitzt, welche in jedem Outbranching enthalten sind. Sei L' die Menge der Knoten, die im Graphen G' Ausgangsgrad 0 besitzen. L' ist also die Menge aller Blätter und isolierten Knoten im Graphen G'.

Da der Graph G' ein Teilgraph eines Optimums T^* ist und die selbe Knotenmenge V besitzt, gilt $|L'| \geq |L^*|$. G' kann aufgrund seines eventuell fehlenden Zusammenhangs und der Berücksichtigung der isolierten Knoten in L' mehr Blätter als T^* besitzen.

Weil für jedes $v \in V_1$ eine Kante zum Graphen G' hinzugefügt wurde, enthält G' genau $|V_1|$ Kanten. Jedes beliebige Outbranching der Eingabeinstanz G hingegen besitzt auf Grund seiner Baumeigenschaft genau $|V| - 1 = |V_1| + |V_2|$ Kanten. Um aus G' ein Outbranching zu konstruieren, müssen demnach noch genau $|V_2|$ Kanten hinzugefügt werden. Diese werden an maximal $|V_2|$ Knoten aus L' angebunden. Dann ist die Anzahl der Blätter jedes beliebigen Outbranchings mindestens $|L'| - |V_2|$. Also ist

$$|L| \geq |L'| - |V_2| \geq |L^*| - |V_2| = |L^*| - \alpha|L^*|.$$

Damit gilt die Schranke 4.4.

Abbildung 4.4 zeigt die beiden Schranken 4.3 und 4.4 schematisch. Hier wird deutlich, dass die Anzahl der Blätter, die durch den Algorithmus ermittelt wird, immer mindestens so hoch wie die jeweils bessere der beiden Schranken ist. Das schlechteste Ergebnis tritt ein, wenn die beiden Schranken 4.3 und 4.4 dieselbe Anzahl an Blättern liefern, also wenn

$$\frac{\alpha}{3}|L^*| = |L^*|(1 - \alpha) \Leftrightarrow \frac{\alpha}{3} = 1 - \alpha \Leftrightarrow 4\alpha = 3 \Leftrightarrow \alpha = \frac{3}{4}.$$

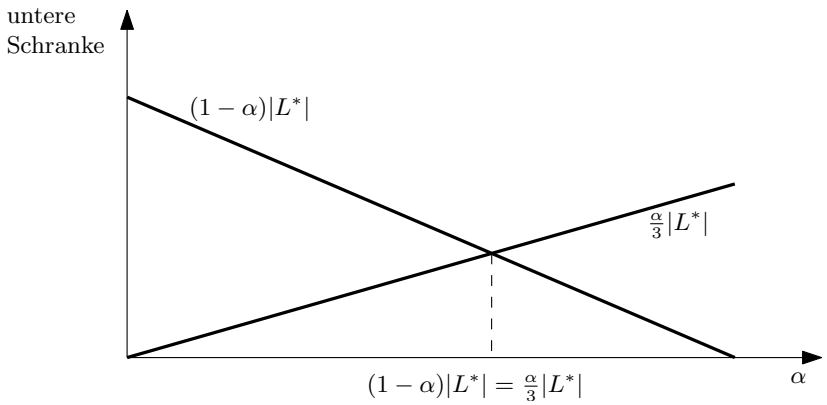


Abb. 4.4.: Untere Schranken für die Anzahl an Blättern, die Algorithmus 6 liefert.

Mit Schranke 4.3 ergibt sich für den Approximationsfaktor f von Algorithmus 6 die Abschätzung

$$f = \frac{|L^*|}{|L|} \leq \frac{|L^*|}{\frac{\alpha}{3}|L^*|} \leq \frac{3}{\alpha} = \frac{3}{\frac{3}{4}} = 4.$$

□

Aus Lemma 4.1, 4.2 und 4.3 folgt Satz 4.2.

Satz 4.2. *Algorithmus 6 ist ein Faktor-4-Approximationsalgorithmus für den azyklischen Fall des RMLO-Problems.*

4.4. Expansionsalgorithmus

In Abschnitt 3.3 wurde ein von Solis-Oba entwickelter Expansionsalgorithmus zur Lösung des MLST-Problems vorgestellt. In Abschnitt 3.4 wurde dieser für gerichtete Graphen modifiziert (vergleiche Algorithmus 3 unter Verwendung von Prozedur 5). Es wurde gezeigt, dass der modifizierte Algorithmus für allgemeine Graphen nicht immer eine zulässige Lösung liefert.

In diesem Abschnitt wird ein Faktor-2-Approximationsalgorithmus entwickelt. Er beruht auf der Expansionstechnik und verwendet Knotenmarkierungen als Statusindikator. Bei der Anwendung des Algorithmus auf einem Graphen $G = (V, E)$ mit Wurzel r werden je Expansionsschritt einerseits Knoten der Eingabeinstanz G in die Lösung übertragen und andererseits bestimmte Knoten in G markiert. Betrachte einen Wald F, der durch mehrere aufeinander folgende Expansionsschritte gewachsen ist. Ein Knoten $v \in V \setminus \{r\}$ ist genau dann in G markiert, wenn sein Eingangsgrad im aktuell gewachsenen Wald F $\deg^-(v) = 1$ ist. Umgekehrt hat ein in G unmarkierter Knoten noch keinen Vorgänger in F zugewiesen bekommen.

Der Algorithmus ist in Algorithmus 10 dargestellt. Auffallend ist, dass er mit einer topologischen Sortierung arbeitet. Tatsächlich bleiben alle hier angeführten

Algorithmus 10: RMLO-Expansionsalgorithmus-azyklisch(G)

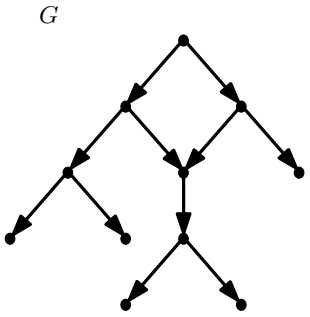
(1) **Input** : azyklischer Wurzelgraph $G = (V_G, E_G)$
(2) **Output** : Outbranching T mit vielen Blättern
(3) Wald $F = (V_F, E_F)$, $V_F \leftarrow \emptyset$
(4) markiere die Wurzel r in G
(5) $F \leftarrow$ Prozedur 11 `// Expansion(`G`)`
(6) $T \leftarrow$ Prozedur 12 `// erstelleOutbranching(`F`)`
(7) **return** T

Prozedur 11 : Expansion(G)

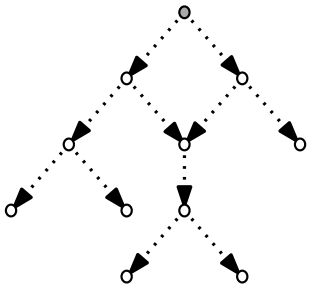
(1) finde eine topologische Sortierung $f : V_G \to \{1, 2, \ldots, |V_G|\}$
(2) **for** $i = 1, \ldots, |V_G|$ *mit* $f(v_i) < \cdots < f(v_{|V_G|})$ **do**
(3) **if** $v_i \notin V_F$ **then**
(4) $V_F \leftarrow V_F \cup \{v_i\}$
(5) **end if**
(6) **if** v_i *besitzt mindestens zwei unmarkierte Nachfolger in* G **then**
(7) füge alle in G unmarkierten Nachfolger von v_i sowie die Kanten, die v_i und seine Nachfolger verbindet, zu F hinzu
(8) markiere die zu F hinzugefügten Nachfolger in G
(9) **end if**
(10) **end for**
(11) **return** F

Prozedur 12 : erstelleOutbranching(F)

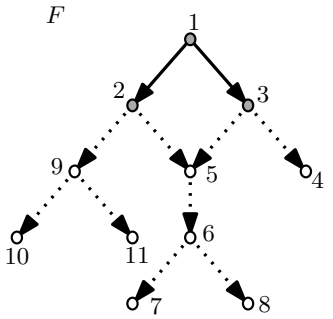
(1) Outbranching $T = (V_T, E_T)$, $V_T \leftarrow V_F$, $E_T \leftarrow E_F$
(2) **foreach** *noch unmarkierten Knoten* v *in* G **do**
(3) wähle $e = (u, v) \in E_G$
(4) $E_T \leftarrow E_T \cup \{e\}$
(5) markiere v
(6) **end foreach**
(7) **return** T



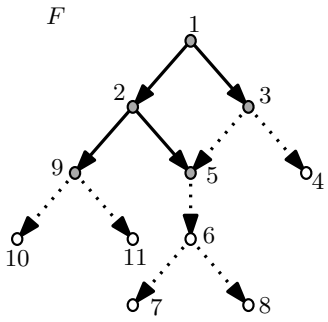
Eingabegraph G



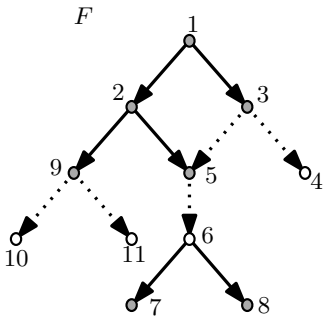
markiere die Wurzel



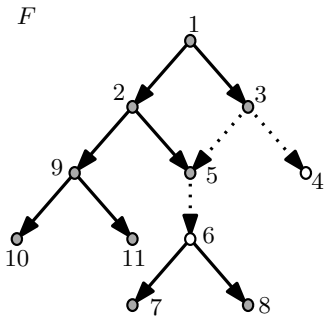
finde topologische Sortierung, expandiere Wurzel, markiere Nachfolger



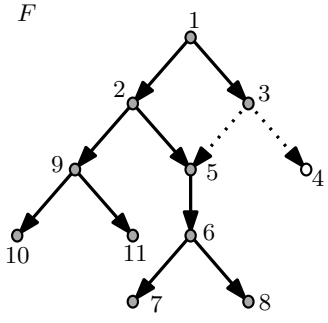
expandiere Knoten $2 < 3$, markiere Nachfolger



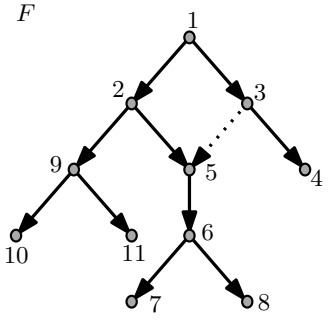
Knoten 3 kann nicht expandiert werden; expandiere Knoten $6 < 9$, markiere Nachfolger



Knoten 9 expandiert, Nachfolger markiert; Wald F besteht aus drei Zusammenhangskomponenten



verbinde unmarkierten Knoten 6, markiere ihn



verbinde unmarkierten Knoten 4, markiere ihn; Outbranching gefunden

Abb. 4.5.: Beispielhafte Anwendung des Algorithmus 10.

Beweise auch ohne die Verwendung einer topologischen Sortierung nahezu unverändert korrekt. Insbesondere ändert sich die Laufzeit nicht. Dennoch wurde die topologische Sortierung hier verwendet, da sie das Expansionskonzept entscheidend vereinfacht. Expansionen können so nur an Blättern durchgeführt werden. Bäume wachsen nach unten. Andernfalls könnten Bäume auch nach oben wachsen.

Abbildung 4.5 verdeutlicht Algorithmus 10 an einem Beispiel. Die Abbildung zeigt unter anderem einen Eingabegraphen G und den von Prozedur 11 konstruierten Wald F. Der Wald F besteht aus zwei nicht-trivialen Bäumen und einem isolierten Knoten. Insbesondere wird deutlich, dass ein Baum nicht in aufeinander folgenden Schritten vollständig expandiert werden muss (im Gegensatz zum Expansionsalgorithmus von Solis-Oba, der Algorithmus 10 inspiriert hat). Obwohl Knoten 2 schon expandiert ist, wird vor der Expansion von Knoten 9 zunächst Knoten 6 erweitert.

Beobachtung 4.1. *Prozedur 11 erstellt einen Wald. Die darin enthaltenen Bäume können auch trivial sein, das heißt, isolierte Knoten.*

Lemma 4.4. *Algorithmus 10 liefert ein Outbranching.*

Beweis. Sei $G = (V_G, E_G)$ mit Wurzel r als Eingabeinstanz gegeben.

Der Algorithmus verwendet die beiden iterativen Prozeduren 11 und 12. In jeder Iteration von Prozedur 11 wird ein Knoten v betrachtet. Dieser Knoten $v \in V_G$ wird zum Wald $F = (V_F, E_F)$ hinzugefügt, falls $v \notin V_F$. Wenn möglich, wird der betrachtete Knoten v expandiert. Im Falle einer Expansion werden weitere Knoten zum Wald F hinzugefügt. Diese sind in G bislang unmarkiert. Sie erhalten Knoten v als eindeutigen Vorgänger in F und werden in G markiert. Knoten v wird nicht markiert. Einem bereits markierten Knoten können keine weiteren Vorgänger hinzugefügt werden. Gesetzte Markierungen können nicht wieder aufgehoben werden.

In jeder Iteration von Prozedur 12 wird genau ein Knoten markiert. Es ist zu beachten, dass die von Prozedur 12 in Zeile 4 geforderte Kante auf Grund des Zusammenhangs des Eingabegraphen G immer existiert.

Alle Knoten $v \in V_G \setminus \{r\}$, die nicht schon in Prozedur 11 markiert werden, werden in Prozedur 12 markiert.

Der Algorithmus terminiert, weil G nur endlich viele Knoten besitzt.

Bei Terminierung des Algorithmus sind alle Knoten markiert. Das bedeutet, dass jeder Knoten der Eingabeinstanz auch in der Ausgabeinstanz enthalten ist, und dass jeder Knoten $v \in V_G \setminus \{r\}$ genau einen Vorgänger besitzt. Außerdem ist die Ausgabe azyklisch, da schon die Eingabe azyklisch ist. Aus diesen drei Eigenschaften folgt, dass die Ausgabe ein Outbranching ist. □

Für den folgenden Beweis sei $G = (V, E)$ die Eingabeinstanz, $|V| = n$ die Anzahl der Knoten und $|E| = m$ die Anzahl der Kanten.

Lemma 4.5. *Algorithmus 10 benötigt Laufzeit $\mathcal{O}(n + m)$.*

Beweis. Die topologische Sortierung in Prozedur 11 benötigt eine Laufzeit von $\mathcal{O}(m+n)$.

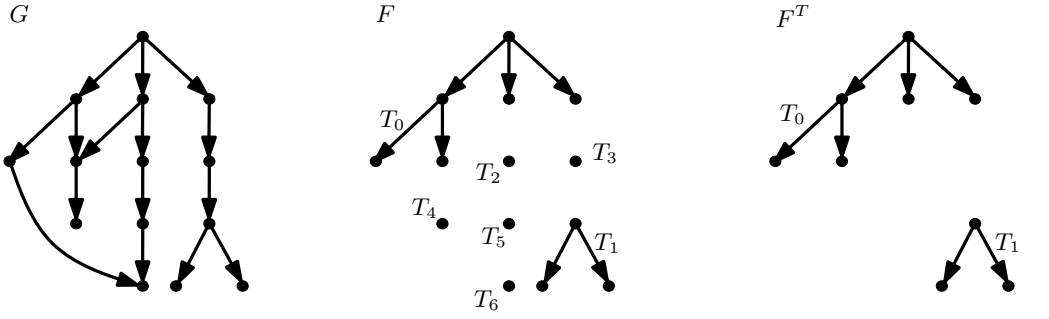
Die Laufzeit der for-Schleife in Zeile 2 von Prozedur 11 benötigt ebenfalls eine Laufzeit von insgesamt $\mathcal{O}(m+n)$. Das Markieren der Knoten, das Einfügen der Knoten und das Einfügen der erforderlichen Kanten benötigt über alle n Durchläufe lineare Laufzeit. Maximal werden alle n Knoten markiert. Es werden genau n Knoten in den Wald F übertragen. Die Anzahl der hinzugefügten Kanten beträgt auf Grund der Baumeigenschaft maximal $n-1$. In jedem Durchlauf i werden außerdem alle Nachfolger des aktuell bearbeiteten Knoten v_i bestimmt. Weiter wird geprüft, wieviele dieser Nachfolger in G noch unmarkiert sind. Dazu werden zu jedem Knoten $v \in V$ seine Nachfolger gespeichert. Dann können die Nachfolger von v_i direkt abgefragt werden. Für jeden Knoten $v_i \in V$ werden schließlich $\deg^+(v_i)$ viele Knoten auf Markierung getestet. Die Summe aller Knotengrade in einem Graphen beträgt höchstens $2m$. Damit wird für alle Tests in n Durchläufen insgesamt $\mathcal{O}(m)$ Laufzeit benötigt.

Also benötigt Prozedur 11 eine Laufzeit von $\mathcal{O}(m+n)$.

Prozedur 12 besitzt ebenso lineare Laufzeit. Es werden maximal $n-1$ Kanten eingefügt. Wenn zu jedem Knoten seine Vorgänger gespeichert werden, kann ein solcher in konstanter Laufzeit gefunden werden.

Insgesamt hat der Algorithmus demnach eine Laufzeit von $\mathcal{O}(n+m)$. □

Für die weiteren Beweise werden nun einige Notationen festgelegt. Abbildung 4.6 verdeutlicht sie an einem Beispiel. Sei F der Wald, der in Prozedur 11 erstellt wird. Sei $F^T = \{T_0, \ldots, T_k\}$ derjenige Wald, der durch Löschung von isolierten Knoten aus dem Wald F entsteht. Der Wald F^T besitzt demnach $k+1$ nichttriviale Zusammenhangskomponenten. Es sei $T_i = (V_i, E_i)$ für alle $T_i \in F^T$. Die Wurzel von $T_i \in F^T$ sei mit r_i bezeichnet. $L(T_i)$ bezeichne die Menge aller Blätter von T_i. Die Menge aller Blätter eines optimalen Outbranchings sei mit L^* notiert.



Eingabe G

Prozedur 10 liefert Wald $F = \{T_0, T_1, \ldots T_6\}$

Wald $F^T = \{T_0, T_1\}$ mit 2 Zusammenhangskomponenten

Abb. 4.6.: Beispiel, das Unterschied zwischen Wald F und Wald F^T aufzeigt.

Abbildung 4.7 motiviert Lemma 4.6. Es zeigt beispielhaft das Wachsen eines Bau-

mes T_i und die Veränderung der Blattanzahl. Es wird deutlich, dass im schlechtesten Fall die Anzahl der Blätter in einem Schritt um genau 1 wächst, wobei die Anzahl der Knoten um genau 2 wächst. Besitzt der expandierte Knoten v Grad $\deg(v)^+ > 2$, so ist das Verhältnis zwischen der Erhöhung der Blattanzahl und Erhöhung der Knotenanzahl noch besser.

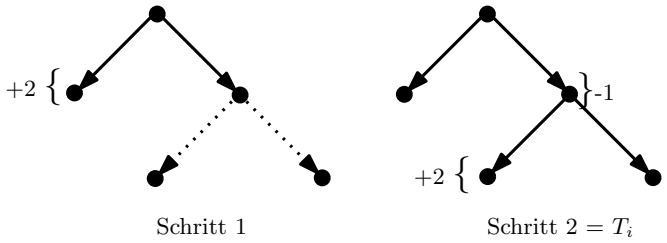


Abb. 4.7.: Baum T_i ist in zwei Schritten gewachsen. Die Anzahl der Blätter wächst jeweils um mindestens 1.

Lemma 4.6. *Algorithmus 10 liefert für jeden Baum $T_i \in F^T$ mindestens $|L(T_i)| \geq (|V_i| + 1)/2$ Blätter.*

Beweis. Dies kann per Induktion über die Expansionsschritte gezeigt werden. Sei $T_i^j = (V_i^j, E_i^j)$ derjenige Teilbaum von T_i, der nach j in T_i durchgeführten Expansionsschritten entstanden ist. In $j = 1$ wird die Wurzel von T_i expandiert.

Induktionsanfang. Nach dem ersten Expansionsschritt besitzt der Baum $T_i \in F^T$ genau eine Wurzel und $|V_i^1| - 1$ Blätter. Mit $|V_i^1| \geq 3$ folgt die Korrektheit der Formel für den Induktionsanfang. Es gilt

$$|L(T_i^1)| = |V_i^1| - 1 \geq \frac{|V_i^1| + 1}{2}.$$

Induktionsschluss. Betrachte die Expansion von T_i^j zu T_i^{j+1}. T_i^{j+1} entsteht, indem l neue Blätter zu T_i^j hinzugefügt werden. Dadurch wird genau ein Blatt zerstört. Das heißt, die Anzahl $|L_i^j|$ der Blätter erhöht sich um $l - 1 \geq 1$ und die Anzahl $|V_i^j|$ der Knoten um l. Es gilt

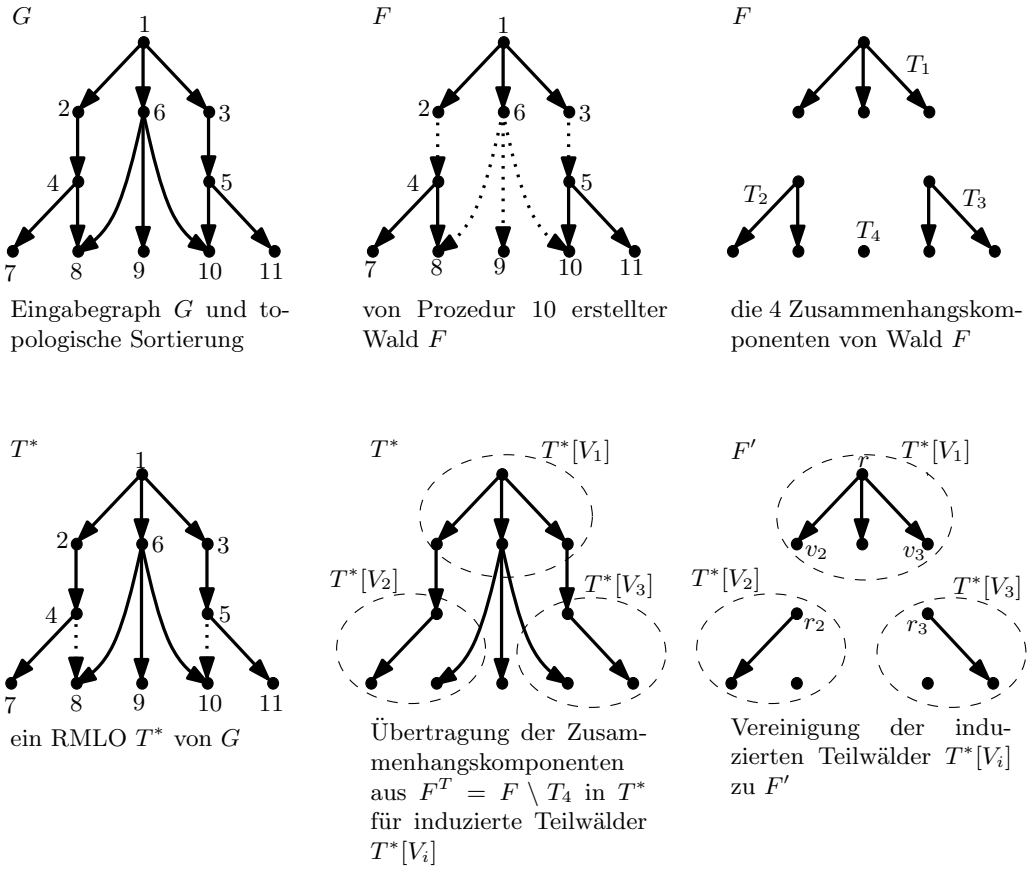
$$\begin{aligned}
|L(T_i^j)| &\geq \frac{|V_i^j| + 1}{2} \text{ (Induktionsanfang)} \\
\Leftrightarrow |L(T_i^j)| + (l-1) &\geq \frac{|V_i^j| + 1}{2} + (l-1) \\
\Leftrightarrow |L(T_i^{j+1})| &\geq \frac{|V_i^j| + 1 + 2l - 2}{2} \\
&= \frac{(|V_i^j| + l) + (l-1)}{2} \geq \frac{|V_i^{j+1}| + 1}{2}.
\end{aligned}$$

□

Lemma 4.7. *Ein RMLO besitzt höchstens* $|L^*| \leq |V(F^T)| - k$ *Blätter.*

Beweis. Sei T^* ein RMLO der Eingabeinstanz G, also ein Outbranching mit maximaler Anzahl an Blättern. Sei F der Wald, der in Prozedur 11 erstellt wird. Der Wald F^T entspricht dem Wald F ohne triviale Zusammenhangskomponenten. Ohne Einschränkung sei T_0 mit Wurzel r_0 der Baum aus F^T, der als erstes expandiert wird.

Zunächst wird zu jedem $T_i \in F^T$ der induzierte Teilwald $T^*[V_i]$ von T^* gebildet. Sei F' die Vereinigung aller induzierten Teilwälder $T^*[V_i]$. Abbildung 4.8 zeigt die Konstruktion eines solchen Teilwaldes F' aus G über F und F^T.



Eingabegraph G und topologische Sortierung

von Prozedur 10 erstellter Wald F

die 4 Zusammenhangskomponenten von Wald F

ein RMLO T^* von G

Übertragung der Zusammenhangskomponenten aus $F^T = F \setminus T_4$ in T^* für induzierte Teilwälder $T^*[V_i]$

Vereinigung der induzierten Teilwälder $T^*[V_i]$ zu F'

Abb. 4.8.: Von der Eingabe zum induzierten Teilwald über die Bäume T_i des Waldes F^T wie vom Beweis zu Lemma 4.7 verwendet.

Die Idee des Beweises ist das Optimum T^* aus dem Teilwald F' (teilweise) zu rekonstruieren. Dafür werden F' einige fehlenden Knoten und Kanten hinzugefügt. Es genügt die Betrachtung der Kanten zur Verbindung der Teilwälder $T^*[V_i] \in F'$ untereinander, einiger isolierter Knoten aus $V \setminus V(F')$ und Kanten zur Anbindung dieser Knoten. Im Folgenden wird gezeigt, dass beim Verbinden der Teilwälder von F' mindestens k Blätter aus F' eine ausgehende Kante erhalten, das heißt,

diese Knoten können im Optimum T^* kein Blatt sein. Zum anderen wird gezeigt, dass die Anbindung jedes Blattes von T^*, welches außerhalb von $V(F')$ liegt (also ein isolierter Knoten in F ist), einen *weiteren* Knoten als Blatt ausschließt. Das heißt, die Anbindung erhöht die Anzahl der Blätter nicht. Aus diesen beiden Fakten folgt die Behauptung.

Um die Behauptung zu beweisen wird für den Knoten r_i jedes Teilwaldes $T^*[V_i] \in F' \setminus T^*[V_0]$ und jeden isolierten Knoten, der Blatt in T^* ist, in eindeutiger Weise jeweils ein Knoten in F' identifiziert, der im Optimum T^* nicht Blatt sein kann. Formaler ausgedrückt, sei L^* die Menge der Blätter von T^*. Identifiziere für jeden Knoten $z \in Z$ mit $Z = \{r_i \mid T^*[V_i] \in F' \setminus T^*[V_0]\} \cup ((V \setminus V(F')) \cap L^*)$ einen Knoten NBK(z), der kein Blatt in T^* ist (*Nicht-Blatt-Knoten*). Ferner soll NBK(z) $\neq$ NBK(z') für $z \neq z'$ mit $z, z' \in Z$ gelten.

Zur Konstruktion des NBK(z) wird vom Knoten z aus der Pfad im RMLO T^* bis zum ersten Knoten $v \neq z$, der in $V(F')$ liegt, zurückverfolgt. Dann ist NBK(z) $= v$. Dieses Vorgehen sei als *Rückwärtssuche* bezeichnet. In Abbildung 4.8 gilt NBK(r_2) $= v_2$ und NBK(r_3) $= v_3$.

Aufgrund der Tatsache, dass der Knoten r_0 im Teilwald $T^*[V_0]$ keine Vorgänger besitzt und damit insbesondere auch keinen Vorgänger in $V(F')$, kann das eben beschriebene Vorgehen zur Auffindung des NBK(r_0) nicht verwendet werden. Tatsächlich ist dies für den Beweis auch nicht notwendig.

Nun soll die paarweise Verschiedenheit der Knoten NBK(z) für alle $z \in Z$ gezeigt werden. Dazu sei angenommen, dass ein Knoten $v \in V(F')$ existiert, sodass $v =$ NBK(z_i) $=$ NBK(z_j) mit $z_i, z_j \in Z$ und $z_i \neq z_j$.

1. Fall: Angenommen die Knoten v, z_i und z_j liegen auf einem gemeinsamen Pfad in T^*. Ohne Einschränkung liege der Knoten z_i auf dem Pfad $P(v, z_j)$ von v nach z_j. Der Fall ist in Abbildung 4.9a dargestellt. Offensichtlich gilt $z_i \in V(F')$. Dann hätte die Rückwärtssuche spätestens bei z_i stoppen und NBK(z_j) $\neq v$ gelten müssen.

2. Fall: Angenommen die Knoten v, z_i und z_j liegen nicht auf einem gemeinsamen Pfad in T^*. Demnach gibt es zwei Pfade $P_i = P(v, z_i)$ und $P_j = P(v, z_j)$ in T^* sowie einen Knoten v_{split}, an denen sich die beiden Pfade P_i und P_j trennen. Sei v_i der Nachfolger von v_{split} in P_i und v_j der Nachfolger von v_{split} in P_j. Der Fall ist in Abbildung 4.9b dargestellt. Da v_{split} nicht expandiert wurde, ist mindestens einer seiner Nachfolger markiert. Sei ohne Einschränkung v_i der markierte Knoten. Es ist $v_i \neq z_i$, da z_i in F^T nicht markiert ist. Es gilt aber $v_i \in V(F^T)$, das heißt, v_i wäre auf der Rückwärtssuche von z_i vor v erreicht worden und demnach NBK(z_i) $\neq v$.

Da alle NBK(z) für $z \in Z$ paarweise verschieden sind und es mindestens k Zusammenhangskomponenten in F^T gibt, folgt die Behauptung. □

Korollar 4.1. *Algorithmus 10 besitzt eine Güte von 2.*

Beweis. Sei OPT die optimale Anzahl der Blätter einer Eingabeinstanz G und LSG die Anzahl der Blätter der Lösung, die der Expansionsalgorithmus ausgibt, wenn er auf G angewendet wird.

Mit Lemma 4.6 und 4.7 gilt

$$OPT \leq |V(F^T)| - k \quad \text{sowie} \quad LSG \geq \sum_{i=0}^{k} \frac{|V_i| + 1}{2} - k.$$

Die Subtraktion von k ergibt sich aus der Tatsache, dass für $k+1$ Zusammenhangskomponenten genau k Pfade benötigt werden, um diese von einem Wald zu einem Baum zusammenzufügen. Dabei werden maximal k Blätter zerstört.

$$\begin{aligned}
\frac{OPT}{LSG} &\leq \frac{|V(F^T)| - k}{\sum_{i=0}^{k} \frac{|V_i|+1}{2} - k} \\
&= \frac{2(|V(F^T)| - k)}{\sum_{i=0}^{k} (|V_i| + 1) - 2k} \\
&= \frac{2(|V(F^T)| - k)}{|V(F^T)| + k + 1 - 2k} \\
&= \frac{2(|V(F^T)| - k)}{|V(F^T)| - k + 1} \\
&\leq \frac{2(|V(F^T)| - k)}{|V(F^T)| - k} = 2
\end{aligned}$$

□

Aus Lemma 4.4, 4.5 und Korollar 4.1 folgt Satz 4.3.

Satz 4.3. *Algorithmus 10 ist ein Faktor-2-Approximationsalgorithmus für den azyklischen Fall des RMLO-Problems.*

Abbildung 4.10 zeigt ein scharfes Beispiel für die 2-Approximation, welches beliebig erweiterbar ist. Der Graph ist so aufgebaut, dass der Teilbaum, der an Knoten 2 hängt, ein vollständiger Binärbaum ist. Jeder Knoten in diesem vollständigen Binärbaum (ausgenommen der Wurzel) besitzt genau zwei eingehende Kanten. Alle Kanten, die in verschiedene Knoten derselben Ebene des Binärbaums eingehen und nicht selbst Teil des Binärbaums sind, besitzen denselben Startknoten. Diese Vorgänger sind für jede Ebene unterschiedlich. Sie seien als *Seitenknoten* bezeichnet. Abbildung 4.10 zeigt den Baum für $k = 3$. Dann besitzt der Binärbaum vier Ebenen (die Wurzel liegt auf Ebene 0).

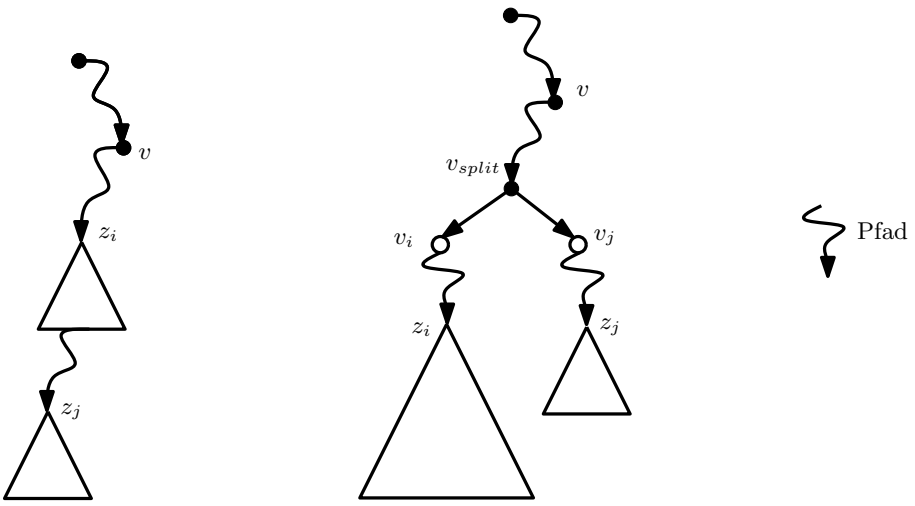


Abb. 4.9.: Es gibt zwei verschiedene Möglichkeiten der Lage der Knoten v, z_i, z_j.

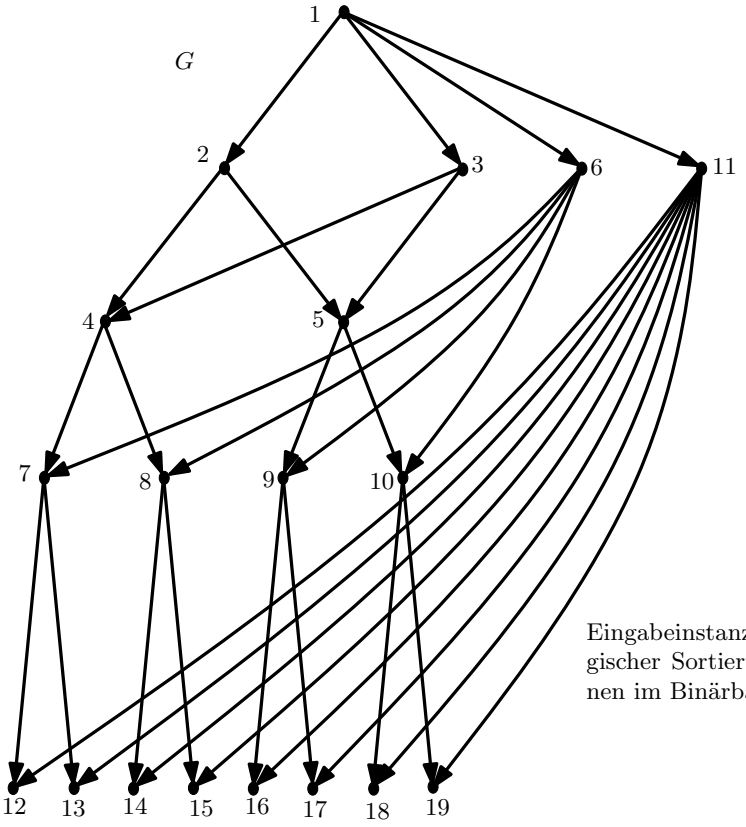
Die topologische Sortierung ist so gewählt, dass der Binärbaum als Teilbaum in der approximierten Lösung enthalten ist.

Der Binärbaum besitzt auf unterster Ebene 2^k Blätter. Diese sind auch Blätter der Lösung. Außerdem sind die k Seitenknoten ebenfalls Blätter. Algorithmus 10 liefert also $2^k + k$ Blätter.

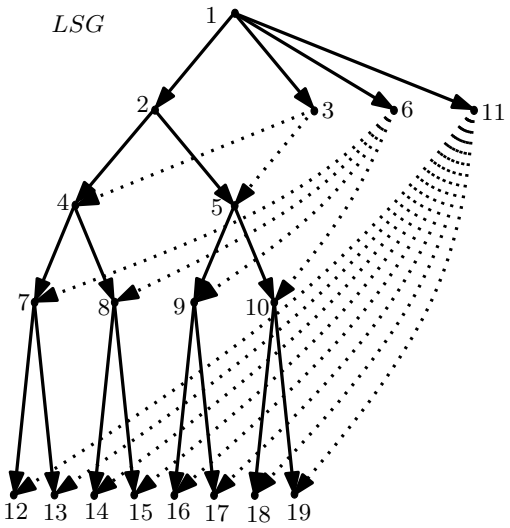
Im Optimum hingegen ist jeder Knoten des Binärbaumes ein Blatt. Ein vollständiger Binärbaum besitzt genau $2^{k+1} - 1$ Knoten.

Sei OPT die optimale Anzahl an Blättern und LSG die Anzahl der Blätter, welche Algorithmus 10 generiert. Dann gilt für beliebig große k

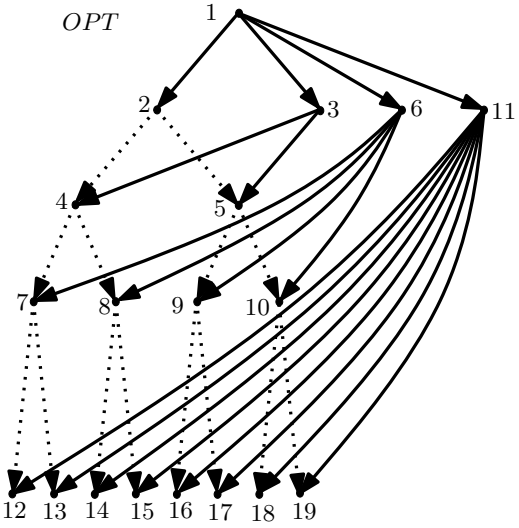
$$\lim_{k\to\infty} \frac{OPT}{LSG} = \lim_{k\to\infty} \frac{2^{k+1}-1}{2^k+k} = \lim_{k\to\infty} \frac{2^1 - \frac{1}{2^k}}{1 + \frac{k}{2^k}} = 2.$$



Eingabeinstanz G mit topologischer Sortierung, k = 3 Ebenen im Binärbaum



Lösung LSG mittels Algorithmus, 11 Blätter



optimale Lösung OPT, 15 Blätter

Abb. 4.10.: Scharfes Beispiel für Algorithmus 10.

5. Fazit

In diesem abschließenden Kapitel sollen die wesentlichen Ergebnisse dieser Arbeit zusammengefasst werden. Zudem wird als Ausblick eine kleine Sammlung offener Probleme gegeben.

5.1. Zusammenfassung

In dieser Arbeit wurde das Problems Rooted Maximum Leaf Outbranching (kurz RMLO-Problem) behandelt. Das heißt, in einem gerichteten Graphen mit ausgezeichneter Wurzel r soll ein gerichteter Spannbaum mit Wurzel r (genannt Outbranching) gefunden werden, welcher die Anzahl der Blätter maximiert (für eine genauere Definition siehe Abschnitt 1.2). Das RMLO-Problem auf allgemeinen Graphen ist NP-schwer [BG09]. Damit kann davon ausgegangen werden, dass für das RMLO-Problem kein exakter effizienter Algorithmus existiert. Aus diesem Grund wurde der Fokus dieser Arbeit auf Approximationsalgorithmen mit beweisbarer Güte gelegt. Der bisher beste bekannte Approximationsalgorithmus, der das allgemeine RMLO-Problem löst, besitzt eine Güte von 92 [DT09]. Für spezielle Graphklassen sind bisher keine Approximationsalgorithmen vorgestellt worden.

Das Forschungsziel (vergleiche Abschnitt 1.5) war einerseits herauszufinden, ob bekannte Approximationsalgorithmen, die das RMLO-Problem in ungerichteten Graphen lösen, auch für das allgemeine RMLO-Problem anwendbar sind. Zudem sollten Algorithmen entworfen werden, die das RMLO-Problem in speziellen Graphklassen approximieren. Andererseits sollten die Algorithmen für diese Klasse bessere Ergebnisse als Faktor 92 liefern.

In Kapitel 3 wurden bekannte Algorithmen zur Lösung des RMLO-Problems im ungerichteten Fall vorgestellt. Anschließend wurde versucht diese auf den gerichteten Fall anzuwenden. Sie haben sich als nicht zielführend erwiesen. Verschiedene Beispiele zeigten, dass eine Übertragung der Algorithmen nicht ohne Weiteres möglich ist. Die ausgegebenen Lösungen sind beliebig schlechte Approximationen oder sogar nicht zulässig.

Als spezielle Graphklasse wurden azyklische Graphen betrachtet. Auch für azyklische Graphen ist das RMLO-Problem NP-schwer [AFG$^+$09]. Außerdem existiert kein PTAS (vergleiche Abschnitt 4.1 und 4.2).

Es wurden zwei Algorithmen mit Faktor 4 beziehungsweise Faktor 2 entworfen. Diese sind damit signifikant besser als der bisher beste bekannte Algorithmus zur Lösung des allgemeinen RMLO-Problems mit Güte 92.

In Abschnitt 4.3 wurde der 4-Approximationsalgorithmus vorgestellt. Der Algorithmus geht von einem Lemma von Daligault und Thomassé [DT09] aus. Seine

Kernaussage ist, dass Graphen, die viele Knoten mit mindestens zwei eingehenden Kanten besitzen, Outbranchings mit vielen Blättern besitzen. Der zugehörige Beweis von Daligault und Thomassé kann in einen Greedy-Algorithmus umgeformt werden.

Die zweite Komponente des Algorithmus besteht in der Beobachtung, dass für Outbranchings von Graphen, die viele Knoten mit nur genau einer eingehenden Kante besitzen, die Struktur des Outbranchings in weiten Teilen schon festgelegt ist. In diesem Fall führen schon beliebige Outbranchings zu guten Ergebnissen.

Der 2-Approximationsalgorithmus in Abschnitt 4.4 ist vom Approximationsalgorithmus zur Lösung des ungerichteten RMLO-Problems von Solis-Oba inspiriert. Der Algorithmus von Solis-Oba wurde bereits in Abschnitt 3.3 vorgestellt und in Abschnitt 3.4 auf den allgemeinen Fall des RMLO-Problems übertragen. Es stellte sich aber heraus, dass diese intuitive Lösung nicht zielführend ist. Zur Lösung des azyklischen RMLO-Problems wurden daher noch einige Änderungen vorgenommen. Die grundlegende Änderung im Vergleich zum Algorithmus aus Abschnitt 3.4 ist die Verwendung von Knotenmarkierungen als Statusindikator. Des Weiteren wurden noch einige wenige Vereinfachungen vorgenommen. Zum Abschluss des Abschnitts wird gezeigt, dass die Güte von 2 eine scharfe Schranke darstellt.

Der Abschnitt 4.4 stellt damit den Kern der Arbeit dar. Der neue Expansionsalgorithmus besitzt dieselbe Güte wie der Expansionsalgorithmus von Solis-Oba. Tatsächlich ist der hier vorgestellte Expansionsalgorithmus einfacher als der von Solis-Oba. Der neue Expansionsalgorithmus verwendet nur eine der drei von Solis-Oba verwendeten Regeln.

Der Algorithmus von Solis-Oba hat nicht nur die Entwicklung von Algorithmus 3 inspiriert, sondern auch den Beweis des Approximationsfaktors. Im direkten Vergleich stellt sich auch hier heraus, dass sich der Beweis der Güte des neuen Expansionsalgorithmus weitaus einfacher gestaltet.

Es ist nicht selbstverständlich, dass ein Problem im azyklischen gerichteten Graphen einfacher zu lösen ist als dasselbe Problem im ungerichteten Graphen. Zum Vergleich wurde schon in der Einführung zu Kapitel 4 das Problem Steiner Tree (kurz ST-Problem) betrachtet. Dieses besitzt einen konstanten Approximationsfaktor $\alpha < 2$ im ungerichteten Fall [RZ00]. Der azyklische gerichtete Fall hingegen ist nicht besser als mit Faktor $\Omega(\log n)$ approximierbar, wobei n die Anzahl der Knoten der Eingabeinstanz ist [Zel97]. Das azyklische gerichtete ST-Problem ist damit komplizierter als das ungerichtette.

Diese Arbeit belegt also, dass es auch Spannbaumprobleme gibt, bei denen der azyklische gerichtete Fall sowohl einfacher als der allgemeine gerichtete Fall als auch einfacher als der ungerichtete Fall ist.

5.2. Offene Probleme

Diese Arbeit stellt keinen Algorithmus zur Lösung des RMLO-Problems in allgemeinen Graphen vor. Dies ist damit ein offenes Problem. Jedoch wurde bereits

in Abschnitt 3.5 eine weitere Idee zur Lösung des Problems skizziert. Bei diesem Algorithmus ist die korrekte Behandlung von Gelenkpunkten essentiell.

Die Expansion in Algorithmus 10 wird vorgenommen, wenn die Blattanzahl mindestens um 1 wächst. Eine offene Frage ist wie sich der Approximationsfaktor verbessert, wenn der Algorithmus in $k > 1$ Phasen abläuft. In der ersten Phase werden nur diejenigen Expansionen durchgeführt, die mindestens k neue Blätter liefern. In der k-ten Phase werden schließlich die Expansionen durchgeführt, die genau ein Blatt liefern.

Wie schon in Abschnitt 1.3 erwähnt, waren bisher keine Approximationsalgorithmen für das RMLO-Problem in speziellen Graphklassen bekannt. In dieser Arbeit wurden zwei Approximationsalgorithmen entwickelt, die das azyklische RMLO-Problem lösen. Auch planare Graphen liefern häufig interessante Ergebnisse. Eine Untersuchung dieser Graphklasse würde sich anbieten.

Abschnitt 4.2 zeigt zwar, dass das azyklische RMLO-Problem nicht beliebig gut approximierbar ist, gibt jedoch keine Antwort auf die Frage nach der unteren Schranke der Güte. Insbesondere ist nichts über die Approximierbarkeit des allgemeinen RMLO-Problems bekannt.

Zuletzt wäre eine experimentelle Evaluierung der gefundenen theoretischen Ergebnisse erstrebenswert. Insbesondere für Algorithmus 6 konnte keine scharfe Schranke nachgewiesen werden. Es besteht Hoffnung, dass dieser eine bessere Güte als 4 aufweist. Auch eine neue theoretische Analyse des Approximationsfaktors wäre denkbar – oder die Konstruktion eines scharfen Beispiels.

A. Glossar

In dieser Arbeit werden einige spezielle, in der Praxis selten genutzte oder unbekannte Begriffe, verwendet. Diese werden stets bei ihrem ersten Vorkommen definiert. Um unnötigen Suchaufwand zu vermeiden, ist hier eine Übersicht aller speziellen Definitionen sowie ihr erstes Auftreten aufgelistet. Alle Begriffe, die nicht vom allgemeinen Gebrauch abweichen, sind in Abschnitt 1.1 aufgelistet.

F^T 43
Prozedur 11 des Algorithmus 10 (vergleiche Seite 40) erstellt aus dem Graphen G einen Wald F aus trivialen und nicht-trivialen Komponenten, das heißt, aus isolierten Knoten und Bäumen T_i. Der Wald F^T ist der Wald, der aus den nicht-trivialen Komponenten $T_0, \ldots, T_k$ von F besteht.

In-d-Knoten 13
Ein *In-d-Knoten* ist ein Knoten, der genau d eingehende Kanten besitzt. Ein *Out-d-Knoten* ist analog definiert.

In-d^*-Knoten 13
Ein Knoten wird *In-d^*-Knoten* genannt, wenn er mindestens d eingehende Kanten besitzt. Analog werden Knoten mit mindestens d ausgehenden Kanten als *Out-d^*-Knoten* bezeichnet.

k-Change 17
Gegeben sei ein Graph G und ein Spannbaum T von G. Ein *k-Change* überträgt genau k Kanten vom Graphen G, welche bisher nicht in T enthalten sind, in den Spannbaum T. Gleichzeitig löscht er genau k Kanten aus T, sodass wieder ein Spannbaum entsteht. Der Tausch wird nur durchgeführt, wenn die Anzahl der Blätter durch ihn wächst.
Ein k-Change im Outbranching ist analog definiert.

MLST 11
In einem ungerichteten Graphen G, heißt ein Spannbaum mit maximaler Blattanzahl auch *MLST* (kurz für *Maximum Leaf Spanning Tree*) von G.

MLST-Problem 11
Beim Maximum Leaf Spanning Tree (kurz *MLST-Problem*) handelt es sich

um das Problem einen MLST in einem gegebenen ungerichteten Graphen zu bestimmen.

Outbranching 10
In einem gerichteten Graphen G mit Wurzel r, heißt ein gerichteter Spannbaum mit Wurzel r auch *Outbranching* von G.

Out-d-Knoten 13
Ein *Out-d-Knoten* ist ein Knoten, der genau d ausgehende Kanten besitzt. Ein *In-d-Knoten* ist analog definiert.

Out-d^*-Knoten 13
Ein Knoten wird *Out-d^*-Knoten* genannt, wenn er mindestens d ausgehende Kanten besitzt. Analog werden Knoten mit mindestens d eingehenden Kanten als *In-d^*-Knoten* bezeichnet.

RMLO 10
Ein *RMLO* (kurz für *Rooted Maximum Leaf Outbranching*) ist ein Outbranching mit maximaler Blattanzahl.

RMLO-Problem 10
Beim Rooted Maximum Leaf Outbranching (kurz *RMLO-Problem*) handelt es sich um das Problem ein Outbranching mit maximaler Anzahl an Blättern in einem gerichteten Graphen zu bestimmen.

Literaturverzeichnis

[AFG+07] Noga Alon, Fedor V. Fomin, Gregory Gutin, Michael Krivelevich und Saket Saurabh: *Parameterized algorithms for directed maximum leaf problems.* In: Lars Arge, Christian Cachin, Tomasz Jurdzinski und Andrzej Tarlecki (Herausgeber): *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, Band 4596 der Reihe *Lecture Notes in Computer Science*, Seiten 352–362. Springer-Verlag, 2007.

[AFG+09] Noga Alon, Fedor V. Fomin, Gregory Gutin, Michael Krivelevich und Saket Saurabh: *Spanning directed trees with many leaves.* SIAM Journal on Discrete Mathematics, 23(1):466–476, 2009.

[BD07] Paul Bonsma und Frederic Dorn: *An FPT algorithm for directed spanning k-leaf.* Computing Research Repository, http://arxiv.org/abs/0711.4052, 2007.

[BF10] Daniel Binkele-Raible und Henning Fernau: *A faster exact algorithm for the directed maximum leaf spanning tree problem.* In: Farid M. Ablayev und Ernst W. Mayr (Herausgeber): *Proceedings of the 5th International Computer Science Symposium in Russia (CSR'10)*, Band 6072 der Reihe *Lecture Notes in Computer Science*, Seiten 328–339. Springer-Verlag, 2010.

[BG09] Jørgen Bang-Jensen und Gregory Z. Gutin: *Digraphs: Theory, Algorithms and Applications*, Kapitel Out-branchings with extremal number of leaves, Seiten 358–363. Springer-Verlag, 2. Auflage, 2009.

[CCC+99] Moses Charikar, Chandra Chekuri, To Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha und Ming Li: *Approximation algorithms for directed Steiner problems.* Journal of Algorithms, 33(1):73–91, 1999.

[DFL+10] Frederic Dorn, Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman und Saket Saurabh: *Beyond bidimensionality: Parameterized subexponential algorithms on directed graphs.* In: Jean Yves Marion und Thomas Schwentick (Herausgeber): *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, Band 5 der Reihe *Leibniz International Proceedings in Informatics (LIPIcs)*, Seiten 251–262. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

[DGKY10] Jean Daligault, Gregory Gutin, Eun Jung Kim und Anders Yeo: *FPT algorithms and kernels for the directed k-leaf problem.* Journal of Computer and System Sciences, 76(2):144–152, 2010.

[DT09] Jean Daligault und Stéphan Thomassé: *On finding directed trees with many leaves.* In: Jianer Chen und Fedor V. Fomin (Herausgeber): *Revised Selected Papers of the 4th International Workshop on Parameterized and Exact Computation (IWPEC'09)*, Band 5917 der Reihe *Lecture Notes in Computer Science*, Seiten 86–97. Springer-Verlag, 2009.

[DV10] Matthew Drescher und Adrian Vetta: *An approximation algorithm for the maximum leaf spanning arborescence problem.* ACM Transactions on Algorithms, 6(3):1–18, 2010.

[GJ79] Michael R. Garey und David S. Johnson: *Computers and intractability: A guide to the theory of NP-completeness.* W. H. Freeman, 1979.

[GMM94] Giulia Galbiati, Francesco Maffioli und Angelo Morzenti: *A short note on the approximability of the maximum leaves spanning tree problem.* Information Processing Letters, 52(1):45–49, 1994.

[KLR08] Joachim Kneis, Alexander Langer und Peter Rossmanith: *A new algorithm for finding trees with many leaves.* In: Seok Hee Hong, Hiroshi Nagamochi und Takuro Fukunaga (Herausgeber): *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC'08)*, Band 5369 der Reihe *Lecture Notes in Computer Science*, Seiten 270–281. Springer-Verlag, 2008.

[LR92] Hsueh I Lu und R. Ravi: *The power of local optimization: Approximation algorithms for maximum-leaf spanning tree.* In: *Proceedings of the Thirtieth Annual Allerton Conference on Communication, Control and Computing*, Seiten 533–542, 1992.

[PY91] Christos H. Papadimitriou und Mihalis Yannakakis: *Optimization, approximation, and complexity classes.* Journal of Computer and System Sciences, 43(3):425–440, 1991.

[RZ00] Gabriel Robins und Alexander Zelikovsky: *Improved Steiner tree approximation in graphs.* In: *Proceedings of the eleventh Annuual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, Seiten 770–779. Association for Computing Machinery, 2000.

[Sol98] Roberto Solis-Oba: *2-Approximation algorithm for finding a spanning tree with maximum number of leaves.* In: Gianfranco Bilardi, Giuseppe F. Italiano, Andrea Pietracaprina und Geppino Pucci (Herausgeber): *Proceedings of the 6th Annual European Symposium on Algorihms (ESA'98)*, Band 1461 der Reihe *Lecture Notes in Computer Science*, Seiten 441–452. Springer-Verlag, 1998.

[Zel97] Alexander Zelikovsky: *A series of approximation algorithms for the acyclic directed Steiner tree problem.* Algorithmica, 18(1):99–110, 1997.