Reduktionsregeln für Baumweite 4

von
Alexander Hein

Bachelorarbeit in Mathematik

vorgelegt der
Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen

im September 2010

angefertigt am Lehrstuhl II für Mathematik
Prof. Dr. Ir. Arie M.C.A. Koster

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

1 Einleitung

1.1 Motivation und Problemstellung

Viele graphentheoretische Probleme liegen in der Komplexitätsklasse NP. Da bis heute nicht bekannt ist ob $P = NP$ ist, werden effiziente Algorithmen für solche Probleme gesucht. Liegen zu einem gegebenen Graphen weitere Informationen vor, so könnte es möglich sein polynomielle Lösungsalgorithmen zu entwerfen. Beispielsweise gibt es für Bäume derartige Algorithmen. Aus diesem Grund wurde in den letzten 30 Jahren das Konzept der Baumweite entwickelt, die beschreibt wie „baum-ähnlich“ ein Graph ist. Dazu wird ein Graph in eine Baumstruktur zerlegt, genannt die Baumzerlegung. Eine äquivalente Charakterisierung der Baumweite ist eine Eliminationsfolge, welche einen Graphen durch Entfernen aller Knoten, unter bestimmten Voraussetzungen, zum leeren Graphen überführt.

Es wurde bereits gezeigt (z.B. in [3], [5] oder [8]), dass viele NP-schwere Probleme, wie etwa HAMILTONIAN CIRCUIT, CHROMATIC NUMBER oder WEIGHTED INDEPENDENT SET, in linearer oder polynomieller Zeit auf Graphen mit beschränkter Baumweite gelöst werden können. Diese benötigen jedoch eine Baumzerlegung oder eine Eliminationsfolge als Input. Die Laufzeit dieser Algorithmen ist aber meist exponentiell in der Baumweite, somit können sie für Graphen mit großer Baumweite unpraktisch sein.

Es wurde versucht, Graphen mit beschränkter Baumweite mithilfe von Reduktionen zu beschreiben. Eine Reduktion überführt einen Graphen in einen kleineren Graphen. Es wurde eine Menge von sechs Reduktionen gefunden (z.B. in [4] oder [9]), mit denen man genau die Graphen mit Baumweite kleiner oder gleich drei in einen leeren Graphen überführen kann. Somit wurden alle Graphen mit Baumweite höchstens drei durch diese Reduktionen charakterisiert. Daniel P. Sanders hat in [16] und [17] diese Arbeit erweitert und eine Menge von Reduktionsregeln gefunden, die Graphen mit Baumweite vier charakterisiert. Weiter hat er einen Algorithmus vorgestellt, der in linearer Zeit eine Eliminationsfolge zu einem Graphen mit Baumweite höchstens vier findet, oder feststellt, dass der Graph eine größere Baumweite hat.

Diese Arbeit lehnt weitestgehend an die Arbeit von Daniel P. Sanders ([16] und [17]) an, weshalb die meisten Definitionen und Sätze von dort übernommen wurden. Ziel dieser Arbeit ist es einen ersten Einblick in die Theorie der Baumweite zu geben und einige wichtige Eigenschaften herzuleiten. Wir werden den Sinn des Reduktions-Prozesses für Graphen mit beschränkter Baumweite verstehen und Reduktionsregeln für Graphen mit Baumweite kleiner oder gleich vier

kennen lernen. Zum Schluss wird ein Algorithmus implementiert und analysiert, der in linearer Zeit entscheidet ob ein Graph Baumweite größer als vier besitzt, oder der eine Eliminationsfolge konstruiert.

1.2 Gliederung der Arbeit

Im letzten Abschnitt von Kapitel 1 werden grundlegende Bezeichnungen und Definitionen behandelt, die aus der Graphentheorie bekannt und fundamental für die Beschreibung von Graphen und Reduktionen sind.

In Kapitel 2 wird zunächst die Baumweite und die damit verbundenen Baumzerlegungen eingeführt und einige grundlegende Eigenschaften diskutiert. Weiter werden zwei weitere Definitionen angegeben, die einer k-Eliminationsfolge und die eines partiellen k-Baumes, und die Äquivalenz zur Baumweite gezeigt, woraus weitere Eigenschaften folgen. Im letzten Abschnittes werden Reduktionen definiert und ihr Eigenschaften diskutiert. Wir werden sehen, dass Minoren und Knoten-Eliminierungen wichtige Hilfsmittel zur Beschreibung von Reduktionen sind.

Kapitel 3 handelt von den in [17] vorgestellten Reduktionsregeln. Zunächst werden die bekannten Reduktionen für Baumweite zwei und drei wiederholt. Danach wird die Menge aller Reduktionen zur Charakterisierung von Graphen mit Baumweite vier aufgestellt, wobei wir einfache Y-Δ Reduktionen, Leiter Reduktionen und Superstrukturen betrachten. Zum Schluss wird gezeigt, weshalb diese Reduktionen Graphen mit Baumweite kleiner oder gleich vier beschreiben.

In Kapitel 4 wird der Algorithmus aus [16] vorgestellt. Zunächst wird das Prinzip des Algorithmus erläutert und anschließend die Korrektheit und die Linearität bewiesen. Weiter wird die Umsetzung des Algorithmus beschrieben und auf mögliche Probleme hingewiesen. Im letzten Abschnitt wird der Algorithmus an partiellen k-Bäumen getestet und die Ergebnisse vorgestellt.

1.3 Grundlegende Definitionen und Bezeichnungen

Ein (endlicher, ungerichteter, einfacher) *Graph* G ist ein geordnetes Paar $G = (V, E)$, wobei V eine endliche Menge von *Knoten* ist und $E \subseteq \binom{V}{2}$[1] eine Menge von *Kanten*. Für einen gegebenen Graphen G bezeichne $V(G)$ die Menge aller Knoten und $E(G)$ die Menge aller Kanten. Wenn der Graph aus dem Kontext klar ist, dann schreiben wir auch V bzw. E anstatt $V(G)$ bzw. $E(G)$.

[1] $\binom{V}{2}$ bezeichnet die Menge aller 2-elementigen Teilmengen von V

Ein *Multigraph* ist ein Tripel $G = (V, E, \varphi)$, wobei (V, E) ein Graph ist und $\varphi : E \to \mathbb{N}^{\geq 1}$ eine Abbildung, welche zählt wie oft eine Kante vorhanden ist. Für einen Multigraphen $G = (V, E, \varphi)$ ist der *zugrunde liegende einfache* Graph (V, E).

Abbildung 1.1 zeigt einen einfachen Graphen (links) und einen Multigraphen (rechts), wobei der einfache Graph zugleich der zugrunde liegende einfache Graph vom Multigraphen ist.

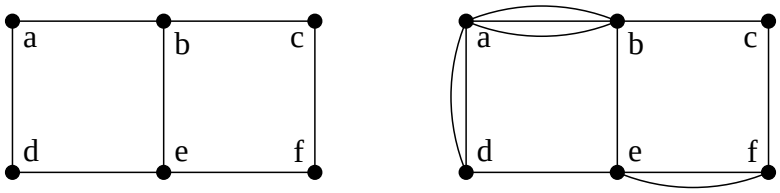

Abbildung 1.1: Ein Multigraph und sein zugrunde liegender einfacher Graph

Im folgenden betrachten wir nur einfache Graphen, wobei alle Definitionen mit offensichtlichen Änderungen auch für Multigraphen übernommen werden können.

Die *Vereinigung* zweier Graphen G und H ist wie folgt definiert; $V(G \cup H) = V(G) \cup V(H)$ und $E(G \cup H) = E(G) \cup E(H)$. Wir nennen einen Graphen H einen *Teilgraph* von einem Graphen G, wenn $V(H) \subseteq V(G)$ und $E(H) \subseteq E(G)$. Der Graph G ist *leer*, falls $V(G) = E(G) = \emptyset$. Der leere Graph wird mit K_0 bezeichnet.

Zwei Graphen G und H heißen *isomorph*, wenn eine bijektive Abbildung $\rho : V(G) \to V(H)$ existiert, sodass folgendes gilt: $\{v, w\} \in E(G) \Leftrightarrow \{\rho(v), \rho(w)\} \in E(H)$. Dann heißt ρ *Isomorphismus von G nach H* und wir schreiben $G \cong H$.

Die Knoten $v, w \in V$ sind *adjazent*, falls $\{v, w\} \in E$. In diesem Fall ist w ein *Nachbar* von v. Die *Nachbarschaft* eines Knoten v ist die Menge aller Nachbarn von v, $N(v) = \{w \in V | \{v, w\} \in E\}$.

Eine Kante $e \in E$ ist *inzident* mit einem Knoten $v \in V$, wenn $v \in e$. Der *Grad* eines Knoten $v \in V$ ist die Anzahl der mit ihm inzidenten Kanten, $deg(v) = |\{\{v, w\} \in E\}|$.

Ein Graph G heißt *vollständig*, wenn $E = \binom{V(G)}{2}$, also wenn zwischen jedem Knotenpaar eine Kante existiert. Für eine Menge von Knoten $W \subseteq V$ bezeichne $G[W] = (W, \binom{W}{2} \cap E)$ den *von W induzierten Teilgraphen* von G. Eine Menge von Knoten $W \subseteq V$ heißt *Clique*, wenn $G[W]$ vollständig ist.

Für einen Graphen G und $v, w \in V$ wird mit $G + vw$ der Graph bezeichnet, der entsteht, wenn man die Kante $\{v, w\}$ zur Kantenmenge E hinzufügt: $G + vw = (V, E \cup \{\{v, w\}\})$ und mit $G - vw$ der Graph bezeichnet, der entsteht, wenn man die Kante $\{v, w\}$ aus der Kantenmenge

E entfernt: $G - vw = (V, E \setminus \{\{v,w\}\})$. Für ein $v \in V$ wird mit $G - v$ der Graph $G[V \setminus \{v\}]$ bezeichnet. Für ein $v \notin V$ wird mit $G + v$ der Graph bezeichnet, der entsteht, wenn man den Knoten v zu V hinzufügt: $G + v = (V \cup \{v\}, E)$.

Für einen Knoten v bezeichnet man mit $G * v$ den Graphen, der entsteht, wenn man in die Menge der benachbarten Knoten von v eine Clique einfügt und anschließend v entfernt. Diese Operation wird *Knoten-Eliminierung* genannt. Formal:

$$V(G * v) = V(G) \setminus \{v\} \text{ und}$$
$$E(G * v) = \left(E(G) \cap \binom{V(G) \setminus \{v\}}{2}\right) \cup \{\{u,w\} | \{v,u\} \in E(G) \text{ und } \{v,w\} \in E(G)\}.$$

Für eine Kante $\{v,w\}$ bezeichnet man mit $G \cdot vw$ den Graphen, der entsteht, wenn man zunächst einen neuen Knoten $u \notin V(G)$ und neue Kanten zwischen u und den Nachbarn von v und w hinzufügt und anschließend die Knoten v und w entfernt. Diese Operation wird *Kanten-Kontraktion* genannt. Formal:

$$V(G \cdot vw) = V(G) \setminus \{v,w\} \cup \{u\} \text{ und}$$
$$E(G \cdot vw) = \left(E(G) \cap \binom{V(G) \setminus \{v,w\}}{2}\right) \cup \{\{u,s\} | \{v,s\} \in E(G) \text{ oder } \{w,s\} \in E(G)\}.$$

Abbildung 1.2 zeigt zu jedem Operator ein Beispiel.

Ein *Weg* (der Länge k) mit Anfangsknoten $v_1 \in V$ und Endknoten $v_k \in V$ ist ein geordnetes k-Tupel $(v_1, \ldots, v_k) \in V^k$ mit $\{v_i, v_{i+1}\} \in E(G)$, für alle $1 \leq i \leq k-1$. Ein *Pfad* ist ein Weg, bei dem alle Knoten verschieden sind. Ein *Kreis* (der Länge k) ist ein Weg $(v_1, \ldots, v_k) \in V^k$ mit $\{v_k, v_1\} \in E(G)$.

Ein Graph heißt *zusammenhängend* , falls es zu allen Knoten $v, w \in V$ einen Weg mit Anfangsknoten v und Endknoten w gibt. Eine *Sehne* ist eine Kante, die mit zwei Knoten eines Kreises inzident ist, aber nicht selbst Kante des Kreises ist.

Ein *Wald* ist ein Graph ohne Kreise. Ein *Baum* ist ein zusammenhängender Wald. Ein *Blatt* eines Baumes oder eines Waldes, ist ein Knoten vom Grad 0 oder 1.

Abbildung 1.3 zeigt einen Pfad, einen Kreis und einen Baum (von links nach rechts).

Für weiter Definitionen siehe z.B. [10]. Damit haben wir unsere Grundlagen geschaffen um Graphen zu beschreiben und können nun im nächsten Kapitel die Baumweite eines Graphen diskutieren.

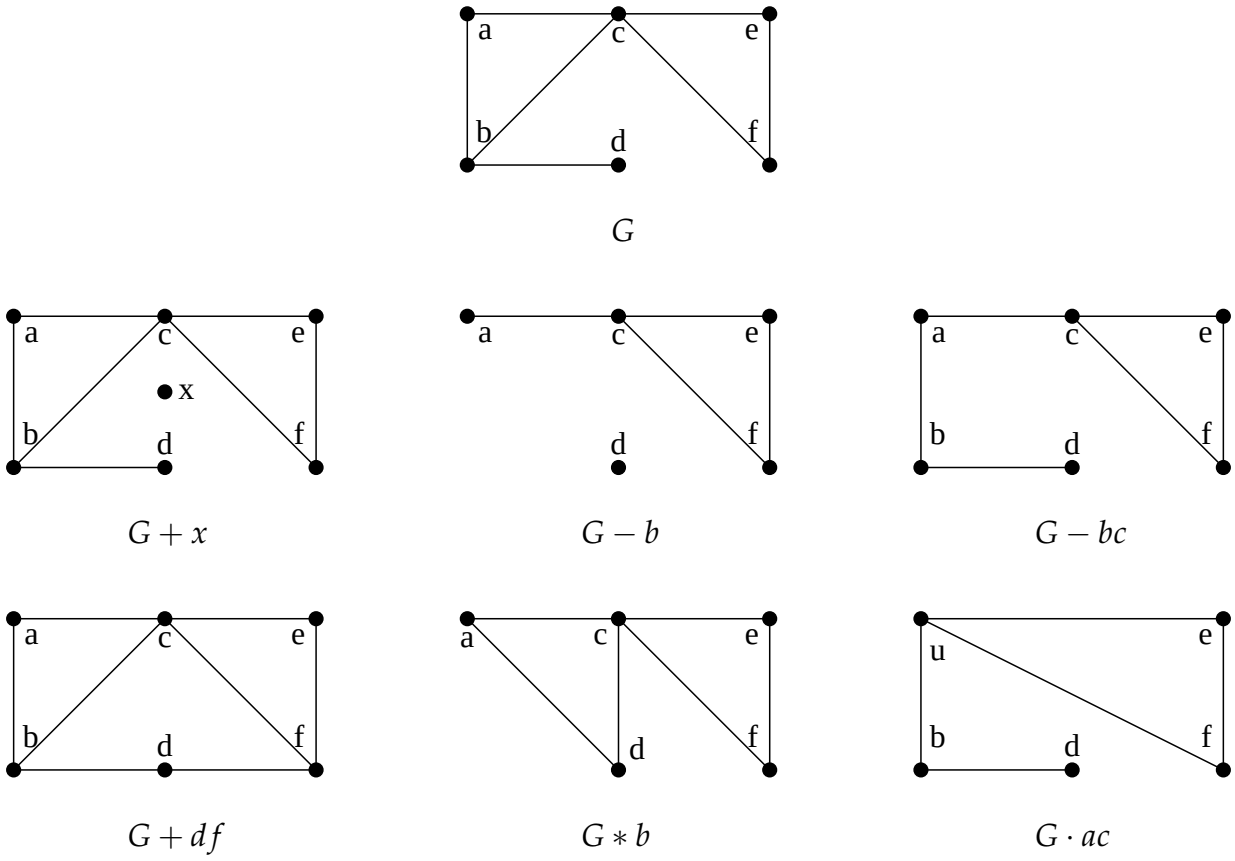

Abbildung 1.2: Beispiele zu verschiedenen Operationen

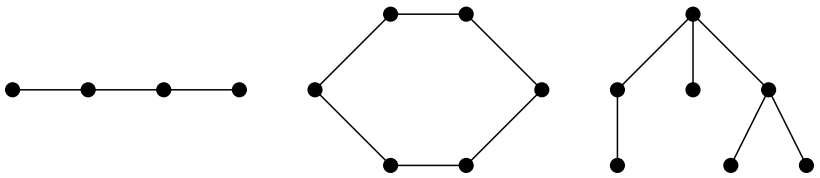

Abbildung 1.3: Beispiel eines Pfades, eines Kreises und eines Baumes

2 Charakteristik von Baumweite und Reduktionen

Dieses Kapitel gibt eine Einleitung in die Theorie der Baumweite. Es werden drei verschiedene Definitionen vorgestellt und erste wichtige Eigenschaften bewiesen. Weiter wird in diesem Kapitel die Anwendung von Reduktionen erläutert und beschrieben, inwieweit diese für unser Ziel relevant sind.

2.1 Baumweite und *k*-Eliminationsfolgen

Die ersten gefundenen Graphen, die eine baum-ähnliche Struktur aufweisen, sind Wälder und series-parallel Graphen [8]. Diese Eigenschaft wurde in folgender Definition generalisiert.

(2.1) Definition: Baumweite
Sei G ein Graph. Eine *Baumzerlegung* eines Graphen G ist ein geordnetes Paar (T, B), wobei T ein Baum ist und $B = \{B_t \subseteq V(G) \mid t \in V(T)\}$ eine Menge von *Taschen*, sodass folgende Bedingungen erfüllt sind:

1. $\bigcup_{t \in V(T)} B_t = V(G)$.
2. Für jedes $\{v, w\} \in E(G)$ gibt es ein $t \in V(T)$ mit $\{v, w\} \subseteq B_t$.
3. Für jedes $v \in V(G)$ ist der von $\{t \in V(T) \mid v \in B_t\}$ induzierte Teilgraph von T ein Baum.

Die *Weite* einer Baumzerlegung (T, B) ist $\max\{|B_t| - 1 \mid t \in V(T)\}$. Die *Baumweite* eines Graphen G ist die minimale Weite einer Baumzerlegung von G und wird mit $\tau(G)$ bezeichnet.

Anschaulich verteilt man die Knoten des Graphen in Taschen, sodass die Taschen eine Baumstruktur aufweisen. Die -1 bei der Definition der Weite ist nur eine Konvention, damit Bäume eine Baumweite von 1 besitzen. Die dritte Bedingung kann man auch so formulieren, dass der induzierte Teilgraph zusammenhängend sein soll. Wir sehen in einem Beispiel einige Baumzerlegungen eines Graphen.

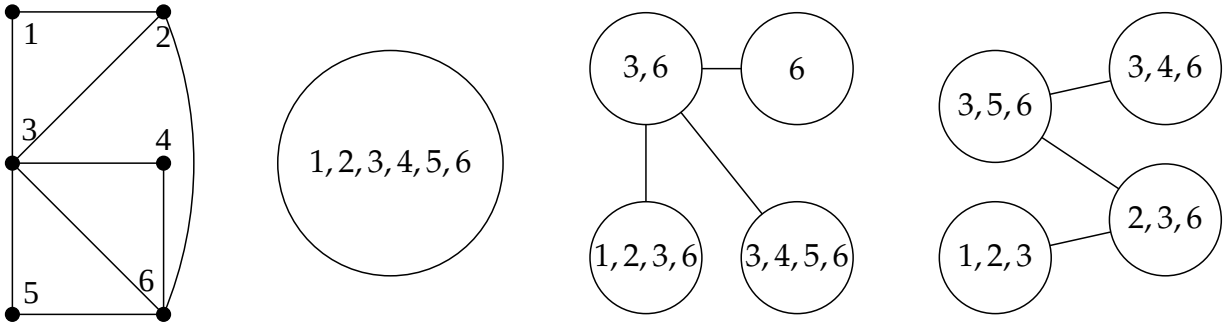

Abbildung 2.1: Beispiel zu Baumweite 2

(2.2) Beispiel
Abbildung 2.1 zeigt einen Graphen und 3 zugehörige Baumzerlegungen. Somit wissen wir, dass die Baumweite des Graphen höchstens 2 ist. Es ist auch leicht zu sehen, dass der Graph keine kleinere Baumzerlegung besitzt, denn sonst wäre er ein Baum.

Ein Grund, warum diese Definition die Baumähnlichkeit eines Graphen gut charakterisiert, ist folgender. Bäume haben die Eigenschaft, dass sie durch entfernen einer Kante in genau zwei Komponenten zerfallen. Baumzerlegungen übertragen diese Eigenschaft auf kleine Knotenmengen. Das folgende Lemma beschreibt diese Eigenschaft.

(2.3) Lemma: [10]
Es sei (T, B) eine Baumzerlegung eines Graphen G, $\{t_1, t_2\} \in E(T)$ und T_1, T_2 die Komponenten von $T - t_1t_2$, mit $t_1 \in T_1$ und $t_2 \in T_2$. Dann trennt $B_{t_1} \cap B_{t_2}$ die Knotenmengen $U_1 = \bigcup_{t \in T_1} B_t$ und $U_2 = \bigcup_{t \in T_2} B_t$ in G (Abbildung 2.2).

Dieses Lemma ist für unser Vorgehen irrelevant, weshalb es auch nicht bewiesen wird. Jedoch verdeutlicht es, weshalb diese Definition sinnvoll ist und weshalb man Baumzerlegungen mit möglichst geringer Weite sucht.

Folgende Lemmata folgen direkt aus der Definition und sind leicht zu beweisen. Sie geben zwei Grundlegende Eigenschaften von Baumzerlegungen an. Diese werden wir im Verlauf häufiger verwenden.

(2.4) Lemma
Wenn G ein Teilgraph von H ist, dann ist $\tau(G) \leq \tau(H)$.

Beweis
Sei (T, B) eine Baumzerlegung von H der Weite k. Es folgt direkt, dass $(T, \bigcup_{t \in V(T)} \{B_t \cap V(G)\})$ eine Baumzerlegung von G ist, mit der Weite höchstens k. $\mathcal{QED}$

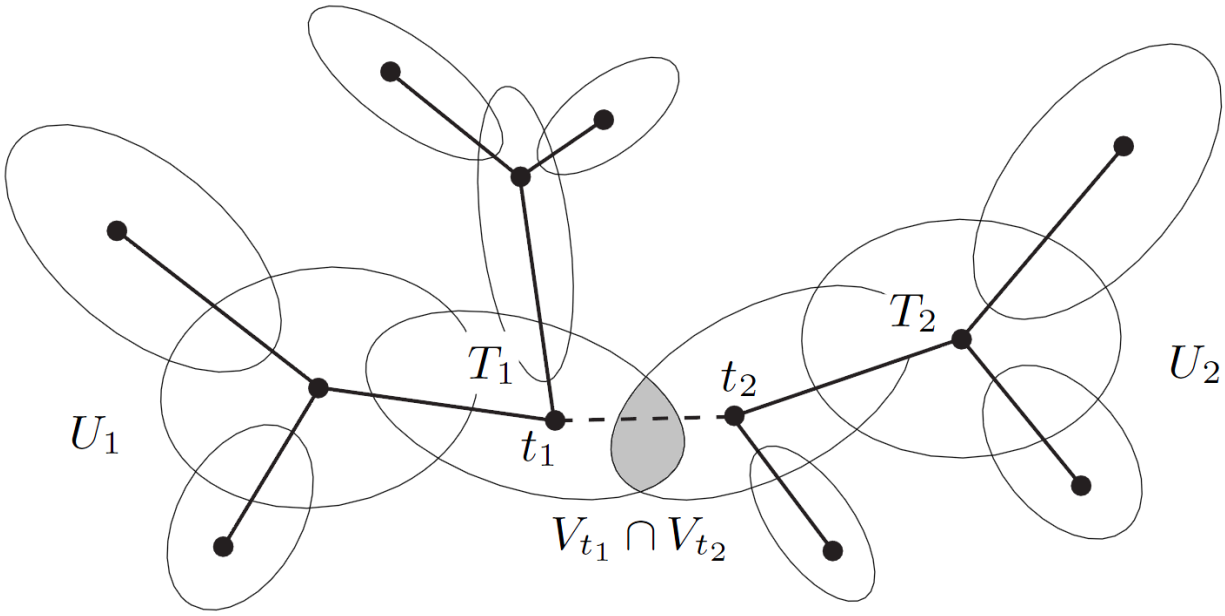

Abbildung 2.2: $V_{t_1} \cap V_{t_2}$ trennt U_1 von U_2 in G [10, S. 280]

(2.5) Lemma

Sei (T, B) eine Baumzerlegung eines Graphen G und $W \subseteq V$ eine Clique von G. Dann existiert ein $t \in V(T)$, sodass $W \subseteq B_t$.

Beweis

Wir führen eine Induktion nach $k = |W|$ durch. Bedingung 1. der Definition einer Baumzerlegung besagt, dass jeder Knoten in einer Tasche enthalten sein muss, und mit Bedingung 2. ist der vollständige Graph auf zwei Knoten in einer Tasche enthalten. Also gilt die Behauptung für $k = 1$ und $k = 2$. Wir nehmen also an, dass die Behauptung für ein festes k gelte. Sei nun W eine Clique von G mit $|W| = k + 1$ und (T, B) eine Baumzerlegung von G. Weiter sei v ein Knoten aus W. Da $W \setminus \{v\}$ eine Clique der Kardinalität k ist, gibt es nach Induktionsvoraussetzung eine Tasche B_t, mit $W \setminus \{v\} \subseteq B_t$. Wenn wir annehmen, dass die Behauptung falsch ist, gibt es zwei Nachbarn u und w von v, so dass $\{v, u\}$ und $\{v, w\}$ in zwei verschiedenen Taschen B_r, B_s liegen, da ansonsten alle Nachbarn von v und der Knoten v selbst in einer Tasche enthalten wären. Mit Bedingung 3. der Definition einer Baumzerlegung folgt dann, dass je ein Weg im Baum T zwischen B_t und B_r, zwischen B_t und B_s und zwischen B_r und B_t existieren muss, was ein Widerspruch zur Kreisfreiheit von T ist. Also gibt es eine Tasche, die W enthält. $\mathcal{QED}$

Mit Hilfe dieses Lemmas können wir im folgenden Beispiel versuchen eine minimale Baumzerlegung zu konstruieren.

(2.6) Beispiel
Wir untersuchen einen ähnlichen Graphen (Abbildung 2.3) wie in Beispiel 2.2. Wenn wir versuchen eine Baumzerlegung der Weite 2 zu konstruieren, erhalten wir wegen Lemma 2.5 zunächst 3 Taschen mit je einer Clique der Größe 3. Diese überdecken alle Kanten, bis auf $\{2,6\}$. Wenn wir die Weite nicht erhöhen wollen, benötigen wir eine weitere Tasche, die diese enthält. Diese 4 Taschen können aber nicht in einer Baumstruktur angeordnet werden (auch nicht unter Zuhilfenahme weiterer Taschen), sodass Bedingung 3. der Definition einer Baumzerlegung erfüllt ist. Also ist die Baumweite des Graphen 3.

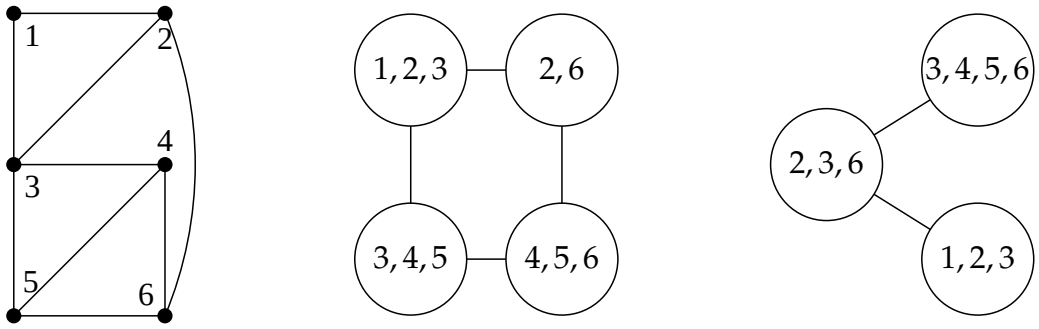

Abbildung 2.3: Beispiel zu Baumweite 3

Da wir Graphen mit beschränkter Baumweite k betrachten wollen (vor allem für $k = 4$), ist folgende Definition nützlich.

(2.7) Definition
Die Menge aller Graphen G mit $\tau(G) \leq k$ wird mit TW_k bezeichnet.

Nun kommen wir wie vorher angesprochen zu einer äquivalenten Charakterisierung von Baumweite, nämlich die einer k-Eliminationsfolge.

(2.8) Definition: *k*-Eliminationsfolge
Sei $G = (V, E)$ ein Graph mit $|V| = n$. Wir nennen eine Folge von Knoten $(v_1, \ldots, v_n) \in V^n$ eine *k-Eliminationsfolge* des Graphen G, falls alle v_i verschieden sind und für jedes i mit $0 \leq i < n$ der Grad vom Knoten v_{i+1} in dem Graphen $G_i = G * v_1 * \cdots * v_i$ höchstens k ist, wobei $G_0 = G$.

Eine k-Eliminationsfolge ist also eine Reihenfolge aller Knoten, die nacheinander durch die Operation $*$ (Knoten-Eliminierung) entfernt werden, wobei jeder Knoten, der entfernt wird, höchstens Grad k hat.

(2.9) Beispiele

Wenn wir die Graphen aus den Beispielen 2.2 und 2.6 betrachten, dann besitzt keiner der Graphen eine 1-Eliminationsfolge, da alle Knoten Grad größer als 1 haben. Zum ersten Graphen kann man jedoch eine 2-Eliminationsfolge finden, wie in Abbildung 2.4 dargestellt wird. Beim

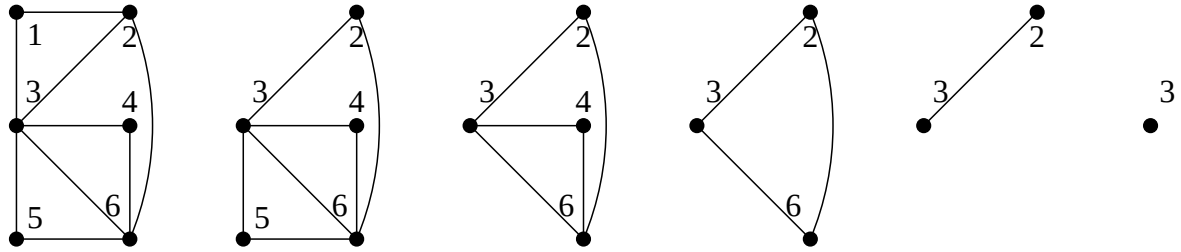

Abbildung 2.4: 2-Eliminationsfolge zum Graphen aus Beispiel 2.2

zweiten Graphen sieht man, dass man den vollständigen Graphen auf 4 Knoten erhält, wenn man 2 Knoten vom Grad 2 eliminiert. Somit besitzt dieser Graph nur eine 3-Eliminationsfolge, wie Abbildung 2.5 zeigt.

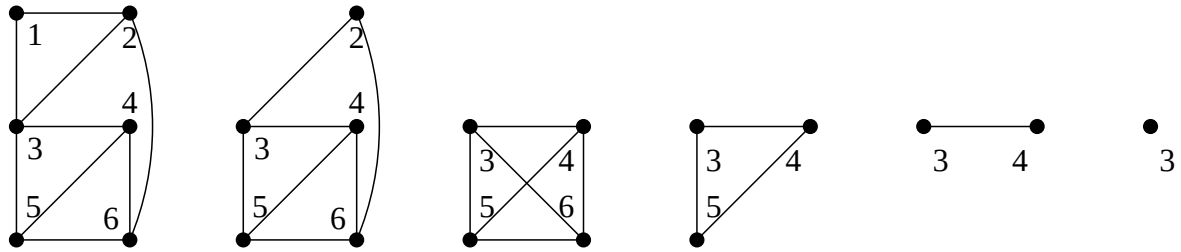

Abbildung 2.5: 3-Eliminationsfolge zum Graphen aus Beispiel 2.6

Diese Beispiele lassen folgenden Satz vermuten, dort wird die Äquivalenz zur Baumweite beschrieben.

(2.10) Satz: Äquivalenz von Baumweite und *k*-Eliminationsfolgen

Ein Graph hat Baumweite höchstens k, genau dann wenn er eine k-Eliminationsfolge besitzt.

Beweis

$\Rightarrow$: In dieser Richtung zeigen wir, dass ein Graph G mit Baumweite höchstens k eine k-Eliminationsfolge besitzt. Sei also (T, B) eine Baumzerlegung von G der Weite k. Wir verwenden eine vollständige Induktion nach $n = |V(T)|$. Falls $n = 1$ ist, dann ist $|V(G)| \leq k+1$ und jede Permutation der Knoten von G ist eine k-Eliminationsfolge. Die Behauptung gelte also für ein beliebiges n. Sei also $|V(T)| = n+1$, $l \in V(T)$ ein Blatt von T und m der eindeutige Nachbar von l. Falls $B_l \subseteq B_m$, dann ist $(T - l, B \setminus \{B_l\})$ eine Baumzerlegung von G der Weite

k und $|V(T)| = n$. Dann folgt die Behauptung nach Induktionsvoraussetzung. Andernfalls sei $B_l \setminus B_m = \{y_1, \ldots, y_j\}$. Nach Definition ist jedes y_i in keiner anderen Tasche außer B_l, für alle $1 \leq i \leq j$. Somit ist die Nachbarschaft von y_i in der Tasche B_l enthalten und $|N(y_i)| \leq k$. Definiert man $H = G * y_1 * \cdots * y_j$, dann ist $(T - l, B \setminus \{B_l\})$ eine Baumzerlegung von H der Weite k und H besitzt nach Induktionsvoraussetzung eine k-Eliminationsfolge. Diese kann durch $y_1, \ldots, y_j$ zu einer k-Eliminationsfolge von G erweitert werden.

$\Leftarrow$: In dieser Richtung konstruieren wir zu einer k-Eliminationsfolge $(v_1, \ldots, v_n)$ eines Graphen G eine Baumzerlegung der Weite höchstens k. Die Graphen G_i sollen wie oben in 2.8 definiert werden, für alle $0 \leq i < n$. Wir konstruiert die Baumzerlegung (T^i, B^i) von G_i rekursiv. G_{n-k-1} besteht nur aus den Knoten $v_{n-k}, \ldots, v_n$, somit ist die Baumzerlegung trivial. Für $0 \leq i < n-k-1$ hat der Knoten v_{i+1} im Graphen G_i einen Grad von höchstens k und dessen Nachbarn bilden eine Clique der Größe höchstens k in G_{i+1}. Sei nun (T^{i+1}, B^{i+1}) eine Baumzerlegung von G_{i+1}. Mit Lemma 2.5 wissen wir, dass es eine Tasche B_t^{i+1} gibt, die alle Nachbarn von v_{i+1} enthält. Sei nun $s \notin V(T^{i+1})$ ein neuer Knoten und $B_s = \{v_{i+1} \cup N(v_{i+1})\}$ eine Tasche, die v_{i+1} und seine Nachbarn enthält, dann ist $(T^i, B^i) = (T^{i+1} + s + st, B^{i+1} \cup \{B_s\})$ eine Baumzerlegung von G_i der Weite höchstens k. $\mathcal{QED}$

Also reicht es aus eine k-Eliminationsfolge zu konstruieren, um zu zeigen, dass ein Graph eine Baumweite kleiner oder gleich k hat. Unser Algorithmus wird solch eine Folge konstruieren, falls sie existiert, um zu zeigen, dass ein Graph Baumweite kleiner oder gleich vier besitzt. In der Speicherung liegt auch der Vorteil dieser Definition. Da es sich nur um eine Permutation der Knoten handelt ist diese sehr leicht zu speichern. Für Baumzerlegungen muss der Inhalt jeder Tasche und die Baumstruktur gespeichert werden, was deutlich höheren Speicherbedarf hat. Jedoch liegt der Vorteil einer Baumzerlegung darin, dass man diese leicht ändern kann. Beispielsweise kann man neue Taschen hinzufügen, oder überflüssige Taschen entfernen. Auch wird erst dadurch die grobe Struktur des Graphen deutlich.

Wie in der Einleitung bereits erwähnt, gibt es Algorithmen, die viele NP-vollständige Probleme in linearer Zeit für einen Graphen lösen, vorausgesetzt es liegt eine Baumzerlegung oder eine k-Eliminationsfolge vor. Unser Algorithmus kann dann die Vorarbeit hierfür verrichten, indem er solch eine Eliminationsfolge findet.

2.2 Partielle k-Bäume und chordale Graphen

In diesem Abschnitt geben wir eine weiter Charakterisierung von Graphen mit beschränkter Baumweite an. Dadurch können wir einige weitere Eigenschaften herleiten. Vor allem ist un-

ser Ziel das Korollar 2.20, dass wir später in vielen Beweisen benötigen. Zunächst führen wir eine neue Klasse von Graphen ein.

(2.11) Definition: *k*-Baum
Die Klasse von *k-Bäumen* ist induktiv wie folgt definiert:

- Ein vollständiger Graph mit k Knoten ist ein k-Baum.
- Ein k-Baum G mit $n+1$ Knoten kann aus einem k-Baum H mit n Knoten konstruiert werden, indem man einen neuen Knoten hinzufügt, der zu genau k Knoten adjazent ist, welche eine k-Clique von H bilden.

(2.12) Bemerkung
Vollständige Graphen auf k und $k+1$ Knoten sind k-Bäume

(2.13) Beispiel
Abbildung 2.6 zeigt wie man einen 3-Baum mit 7 Knoten aus einem vollständigen Graphen auf 4 Knoten konstruieren kann.

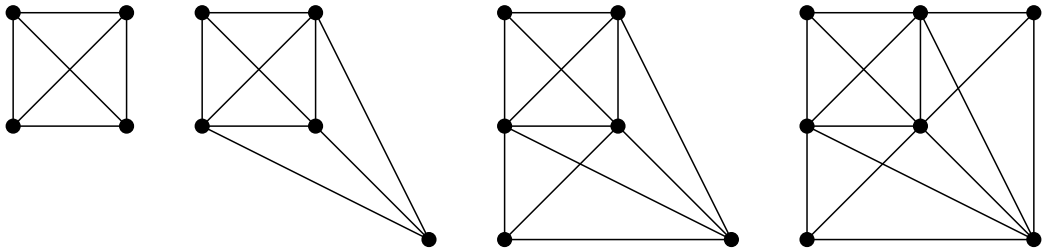

Abbildung 2.6: 3-Bäume mit 4,5,6 und 7 Knoten

Mit dem folgenden Satz können wir die Baumweite von k-Bäumen leicht bestimmen.

(2.14) Satz
Jeder k-Baum hat Baumweite höchstens k.

Beweis
Wir verwenden eine Induktion nach der Anzahl der Knoten n, wobei ein k-Baum mit k Knoten offensichtlich Baumweite $k-1$ hat. Die Behauptung gelte also für ein beliebiges n. Sei nun $G = (V, E)$ ein k-Baum mit $n+1$ Knoten. Nach Definition gibt es ein $v \in V$, sodass

$G[V \setminus \{v\}]$ ein k-Baum ist und die Nachbarn von v eine Clique K der Größe k bilden. Mit der Induktionsvoraussetzung hat $G[V \setminus \{v\}]$ Baumweite höchstens k und eine dazugehörige Baumzerlegung (T, B). Mit Lemma 2.5 gibt es eine Tasche B_t, die alle Nachbarn von v enthält. Sei nun $s \notin V(T)$ und $B_s = K \cup \{v\}$, dann ist $T' = (V(T) \cup \{s\}, E(T) \cup \{s,t\})$ und $B' = B \cup \{B_s\}$ eine Baumzerlegung von G der Weite k. $\mathcal{QED}$

Im folgenden wird eine weitere Klasse von Graphen eingeführt und der Bezug zur Baumweite gezeigt.

(2.15) Definition: Partieller *k*-Baum
Ein *partieller k-Baum* ist entweder ein Graph mit weniger als k Knoten, oder ein Teilgraph eines k-Baumes mit der gleichen Knotenmenge.

(2.16) Satz: Äquivalenz von Baumweite und partiellen *k*-Bäumen
G ist ein partieller k-Baum, genau dann, wenn G eine Baumweite höchstens k besitzt.

Beweis
$\Rightarrow$: In diese Richtung wollen wir zeigen, dass jeder partielle k-Baum Baumweite höchstens k besitzt. Das folgt direkt aus der Definition 2.15, Lemma 2.4 und Satz 2.14.

$\Leftarrow$: In dieser Richtung wollen wir zeigen, dass jeder Graph mit Baumweite höchstens k ein partieller k-Baum ist. Sei also $G = (V, E)$ ein Graph und (T, B) eine Baumzerlegung von G der Weite höchstens k. Wenn wir zeigen, dass ein k-Baum $H = (V, E')$ existiert, sodass für jedes $t \in T$ der Graph $G[B_t]$ Teilgraph einer Clique der Größe $k+1$ von H ist, dann folgt die Behauptung. Dies zeigen wir durch Induktion nach $n = |V|$. Der Fall $n \leq k+1$ folgt entweder aus der Definition 2.15 oder aus Bemerkung 2.12. Wir setzen voraus, dass die Behauptung für ein festes n gilt. Sei nun $n > k+1$ und $t \in V(T)$ ein Blatt von T mit dem eindeutigem Nachbarn $s \in V(T)$. Falls $B_t \subseteq B_s$, können wir das Blatt t aus T entfernen und mit der übrigen Baumzerlegung fortfahren. Sei andernfalls v der einzige Knoten in $B_t \setminus B_s$ (ansonsten kann B_t in Blattknoten aufgeteilt werden, sodass der entstandene Knoten die Bedingung erfüllt). Da v nicht in B_i enthalten ist, für $i \neq t$, müssen alle Nachbarn von v in B_t enthalten sein. Jetzt können wir die Induktionsvoraussetzung auf den Graphen $G[V \setminus \{v\}]$ und die Baumzerlegung $(T[V(T) \setminus \{t\}], B \setminus \{B_t\})$ anwenden und erhalten einen k-Baum H'. Damit ist $G[B_s]$ Teilgraph einer Clique der Größe $k+1$ in H'. Da alle Nachbarn von v (höchstens k Stück) in B_s liegen, induzieren sie einen Teilgraph einer k-Clique C von H' in G. Somit können wir den Knoten v zu der Clique C hinzufügen und erhalten eine $(k+1)$-Clique und damit den k-Baum H. $\mathcal{QED}$

Da die Baumweite von partiellen k-Bäumen durch k beschränkt ist und diese leicht zu konstruieren sind, werden sie uns helfen, den in Kapitel 4 vorgestellten Algorithmus zu testen und zu analysieren.

Ein leichtes Kriterium, welches uns eine untere Schranke für die Baumweite liefert und uns gleichzeitig für festes k eine lineare Schranke für die Anzahl der Kanten liefert, ist folgendes.

(2.17) Lemma
Ein Graph G habe eine Baumweite von höchstens k. Dann gilt $|E(G)| \leq k|V(G)|$.

Beweis
Ein k-Baum hat nach Definition genau $\frac{k(k-1)}{2} + k(|V| - k) = k|V| - \frac{k(k+1)}{2} \leq k|V|$ Kanten. Also hat ein partieller k-Baum höchstens so viele. $\mathcal{QED}$

Jetzt kommen wir zu einer wichtigen Eigenschaft von k-Bäumen. Auch wenn diese nicht auf partielle k-Bäume zutrifft, hilft sie uns Graphen mit beschränkter Baumweite zu beschreiben.

(2.18) Definition: chordal
Ein Graph G ist *chordal*, wenn jeder Kreis der Länge größer als 3 in G eine Sehne besitzt.

(2.19) Satz
k-Bäume sind chordal.

Beweis
Sei ein k-Baum $G = (V, E)$ gegeben. Wir beweisen die Aussage mit vollständiger Induktion nach $|V| = n$, wobei der Induktionsanfang $n = k$ ist. Dann ist G der vollständige Graph auf k Knoten und ist chordal, da er jede mögliche Kante enthält. Die Behauptung gelte also für ein beliebiges n. Für $n > k$ gibt es nach Definition einen Knoten x, dessen Nachbarn eine k-Clique bilden und sodass $G - x$ ein k-Baum ist. Mit der Induktionsvoraussetzung ist $G - x$ chordal. Sei C ein Kreis der Länge mindestens 4. Falls x kein Knoten des Kreises ist, dann hat C eine Sehne, denn $G - x$ ist chordal. Ansonsten seien y, z mit $y \neq z$ zwei Nachbarn von x in C. Dann ist $\{y, z\}$ eine Sehne von C, da die Nachbarschaft von x eine Clique bildet. $\mathcal{QED}$

(2.20) Korollar
Jeder Graph aus TW_k ist ein Teilgraph eines chordalen Graphen in TW_k.

Beweis
Jeder Graph mit Baumweite höchstens k ist Teilgraph eines k-Baumes. Dieser wiederum ist chordal und hat Baumweite höchstens k. $\mathcal{QED}$

Mit Hilfe dieses Satzes wissen wir, dass wenn wir einen Graphen $G \in TW_k$ haben, der einen Kreis (a, b, c, d) besitzt, dann entweder $G + ac \in TW_k$ oder $G + bd \in TW_k$ ist. Dies werden wir in vielen Beweisen in Kapitel 3 benutzen.

Wir wollen uns im folgenden speziell mit Graphen aus TW_4 beschäftigen. Mithilfe von Reduktionen wollen wir ein Verfahren aufstellen, dass die Mitgliedschaft eines Graphen in TW_4 testet.

2.3 Reduktionen

Bevor wir konkrete Reduktionen angeben können, müssen wir erst einmal Reduktionen beschreiben. Wir wollen Teile eines Graphen durch kleinere Teile ersetzen. Diesen Vorgang wollen wir als Reduktion definieren.

(2.21) Definition: Zerlegung
Eine *Zerlegung* eines Graphen G ist ein Tripel $(A, B, (v_1, \ldots, v_k))$, wobei A und B Teilgraphen von G sind mit $E(A) \cup E(B) = E(G)$, $E(A) \cap E(B) = \emptyset$ und $V(A) \cap V(B) = \{v_1, \ldots, v_k\}$.

Mit Hilfe einer Zerlegung können wir einen Graph in zwei Teilgraphen trennen, sodass die Vereinigung dieser wieder den ursprünglichen Graphen ergeben und diese nur eine endliche Menge von Knoten gemeinsam haben.

(2.22) Definition: Struktur
Eine *Struktur* S ist ein Paar $(G(S), (u_1, \ldots, u_j))$, wobei $G(S)$ ein Graph ist und $u_1, \ldots, u_j$ verschiedene Knoten von $G(S)$ sind, genannt die *Ankerknoten* von S. Alle übrigen Knoten von $G(S)$ werden *innere Knoten* von S genannt. Für eine Struktur S definieren wir $V(S) = V(G(S))$ und $E(S) = E(G(S))$.
Zwei Strukturen $S = (G(S), (u_1, \ldots, u_j))$ und $T = (G(T), (v_1, \ldots, v_k))$ heißen *isomorph*, wenn $j = k$ ist und es einen Isomorphismus ρ von $G(S)$ nach $G(T)$ gibt, mit $\rho(u_i) = v_i$ für alle $1 \leq i \leq j$.
Ein Graph G *enthält eine Struktur* S, wenn es eine Zerlegung $(A, B, (v_1, \ldots v_k))$ von G gibt, sodass $(B, (v_1, \ldots v_k))$ isomorph zu S ist. Für einen Knoten $x \in V(G)$ *enthält* G *die Struktur* S *bei* x, wenn G die Struktur S enthält, sodass die isomorphe Kopie von S in G x als einen Knoten besitzt.
Für zwei Graphen G und H und zwei Strukturen S und T ist der Graph H *aus* G *entstanden durch Ersetzen von* S *durch* T, wenn G eine Zerlegung $(A, B, (v_1, \ldots v_k))$ und H eine Zerlegung $(A, C, (v_1, \ldots v_k))$ haben, sodass $(B, (v_1, \ldots v_k))$ isomorph zu S ist und $(C, (v_1, \ldots v_k))$ isomorph zu T ist.

Eine Struktur ist also ein Teilgraph, der zum restlichen Graphen nur über die Ankerknoten verbunden ist. Diese Struktur können wir dann durch eine andere Struktur ersetzen. Bei Reduktionen wird stets eine Struktur durch eine Kleinere ersetzt.

(2.23) Definition: Reduktion
Eine *Reduktion* ist ein Paar von Strukturen (S_R, T_R) mit der gleichen Folge von Ankerknoten und $|V(S_R)| > |V(T_R)|$.
Für Graphen G, H und eine Reduktion R ist der Graph H *aus G entstanden durch Anwendung von R*, wenn H aus G entstanden ist durch Ersetzen von S_R durch T_R.
Für eine Reduktion R definieren wir die Ordnung $H \leq_R G$, wenn es eine Folge von Graphen $H = G_1, \ldots, G_k = G$ gibt, sodass für jedes $1 \leq i < k$, G_i aus G_{i+1} entstanden ist durch Anwendung von R.
Für eine Menge A von Graphen heißt eine Reduktion *A-sicher*, wenn für alle Graphen G, H mit $H \leq_R G$ gilt: $G \in A \Leftrightarrow H \in A$.
Ein Graph G *enthält* eine Reduktion R, wenn G die Struktur S_R enthält und jede Kante zwischen zwei Ankerknoten mit mindestens einem Knoten inzident ist, dessen Bild in G einen Grad von höchstens 6 in G hat[1].
Eine Menge Q von Reduktionen heißt *A-vollständig*, wenn jeder nicht-leere Graph in A ein $R \in Q$ enthält.

Eine Anwendung einer Reduktion auf einen Graphen ist also dass, was man sich intuitiv darunter vorstellt. Man nimmt einen Teil des Graphen, der vom Rest getrennt ist, und ersetzt ihn durch einen kleineren Graphen. Unser Ziel ist es, Graphen mit einer Baumweite kleiner oder gleich vier durch Reduktionen zu charakterisieren. Wir wollen Reduktionen angegeben, die durch Anwenden dieser, die Baumweite von Graphen nicht erhöhen und mit denen man von der Baumweite des kleineren Graphen auf die Baumweite des größeren Graphen schließen kann. Also muss diese Menge TW_4-sicher sein. Wenn diese Menge zudem noch TW_4-vollständig ist, dann charakterisiert diese Menge von Reduktionen genau die Graphen aus TW_4. Denn wir können mit dieser Menge einen Prozess aufstellen, welcher die Mitgliedschaft eines Graphen in TW_4 testet. Für einen Graphen G sucht man eine Reduktion aus dieser Menge. Wenn eine gefunden wurde, dann wendet man diese an und erhält einen Graphen H. Es ist also $H \in TW_4$ genau dann wenn $G \in TW_4$. Jetzt wendet man das Verfahren auf H an u.s.w. Falls in einem Schritt der Graph keine Reduktion besitzt, dann ist er nicht in TW_4 enthalten, da die Menge TW_4-vollständig ist. Somit ist dann der ursprüngliche Graph G nicht in TW_4 enthalten. Ansonsten wird in jedem Schritt eine Reduktion gefunden und, da sich die Anzahl der Knoten verringert, terminiert dieser

[1] Diese Einschränkung ist für den später vorgestellten Algorithmus notwendig und wird in Kapitel 4 erläutert.

Prozess mit dem leeren Graphen K_0. Der Algorithmus aus Kapitel 4 wird genau nach diesem Prinzip arbeiten.

(2.24) Bemerkung
Wenn wir zeigen wollen, dass eine Reduktion R A-sicher ist, und wir zwei Graphen G und H mit $G \leq_R H$ gegeben haben, dann können wir o.B.d.A. annehmen, dass G aus H entstanden ist durch Anwendung von R. Weiter können wir der Einfachheit halber annehmen, dass die Strukturen einer Reduktion gleich anstatt isomorph sind zu den Teilen der Zerlegung von Graphen.

(2.25) Beispiel
Abbildung 2.7 zeigt eine Reduktion $R = (S_R, T_R)$ und drei Graphen G, H und J, wobei G die Reduktion R, bzw. die Struktur S_R enthält und J nicht. Dabei ist der Graph H aus G entstanden durch Anwendung von R. In den Strukturen werden die Ankerknoten schwarz dargestellt und die inneren Knoten weiß.

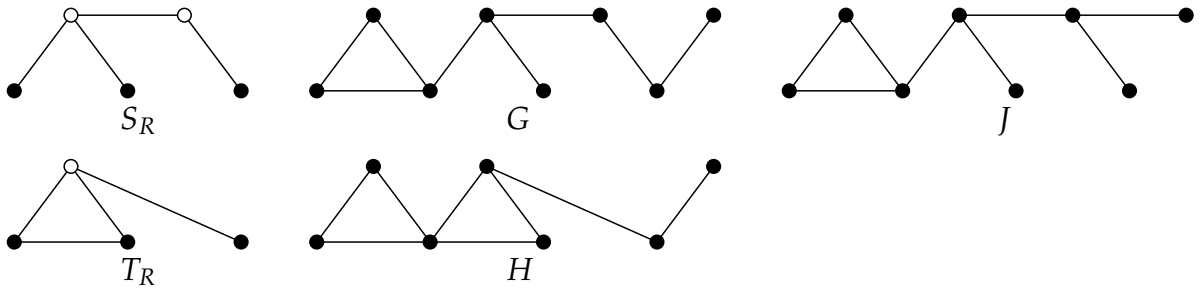

Abbildung 2.7: Beispiel zu Reduktionen

Minoren und Folgen von Knoten-Eliminierungen sind wichtige Hilfsmittel um Reduktionen zu beschreiben. Aus diesem Grund führen wir weitere Bezeichnungen ein.

(2.26) Definition
Ein Graph G ist ein *Minor* des Graphen H, falls G isomorph zu einem Graphen ist, der aus Anwenden endlich vieler Kanten-Kontraktionen auf einen Teilgraphen von H entsteht. Wir definieren für zwei Graphen G und H, $G \leq_m H$, wenn G ein Minor von H ist und $G \leq_* H$, wenn G isomorph zu einem Graphen ist, der aus Anwenden von endlich vielen Knoten-Eliminierungen von Knoten mit einem Grad höchstens 4 auf H entsteht.

(2.27) Definition: 4-Sterne Reduktion
Wir nennen eine Reduktion $R = (S_R, T_R)$ eine *4-Sterne Reduktion*, wenn $G(T_R) \leq_* G(S_R)$.

4-Sterne Reduktionen sind algorithmisch sehr wichtig. Damit kann die Anwendung der Reduktion durch eine einfache Folge von Knoten-Eliminierungen beschrieben werden. Wenn eine Folge von 4-Sterne Reduktionen also einen Graphen in den leeren Graphen überführt, dann ist eine 4-Eliminationsfolge konstruiert, die zeigt, dass der Graph eine Baumweite von höchstens 4 besitzt. Andererseits lässt sich aus Reduktionen, die einen Graphen in den leeren Graphen überführen, aber keine 4-Sterne Reduktionen sind, nur schwer eine 4-Eliminationsfolge bestimmen.

Folgende Lemmata betreffen Aussagen, die die Baumweite von Minoren und Knoten-Eliminierungen beschreiben. Diese Aussagen sind für uns wichtig, da viele Reduktionen darauf zurückgeführt werden können.

(2.28) Lemma
Wenn $G \in TW_k$ und $H \leq_m G$ ist, dann ist auch $H \in TW_k$.

Beweis
Sei o.B.d.A. $H = G \cdot vw$ für $\{v, w\} \in E(G)$. Sei u die Bezeichnung des neuen Knotens in H und (T, B) eine Baumzerlegung der Weite höchstens k von G. Zu B_t mit $t \in V(T)$ definiere $B'_t = B_t$, falls $v, w \notin B_t$, und $B'_t = B_t \setminus \{v, w\} \cup \{u\}$, sonst. Dann ist $(T, \cup_{t \in V(T)} \{B'_t\})$ eine Baumzerlegung von H mit Weite höchstens k, was leicht zu sehen ist. $\mathcal{QED}$

(2.29) Lemma
Wenn $G \in TW_4$ und $G \leq_* H$ ist, dann ist auch $H \in TW_4$.

Beweis
Sei o.B.d.A. $G = H * v$ für ein $v \in V(H)$, wobei der Grad von v höchstens 4 ist. Nach Voraussetzung hat G eine 4-Eliminationsfolge $(v_1, \ldots, v_n)$. Dann ist auch $(v, v_1, \ldots, v_n)$ eine 4-Eliminationsfolge von H, die zeigt, dass H Baumweite höchstens 4 hat. $\mathcal{QED}$

(2.30) Lemma
Wenn $G \leq_R H$ bereits $G \leq_m H$ und $G \leq_* H$ impliziert, dann ist R TW_4-sicher.

Beweis
Wir müssen zeigen, dass $G \in TW_4 \Leftrightarrow H \in TW_4$ gilt. Wenn $G \in TW_4$, dann folgt mit Lemma 2.29, dass $H \in TW_4$ und wenn umgekehrt $H \in TW_4$, dann folgt mit Lemma 2.28, dass $G \in TW_4$. Also gilt die Äquivalenz. $\mathcal{QED}$

Mit Hilfe dieser Lemmata können wir die Sicherheit der meisten Reduktionen im nächsten Kapitel beweisen. Dort wollen wir eine TW_4-vollständige Menge von TW_4-sicheren 4-Sterne Reduktionen aufstellen. Wir können aber nicht hoffen, dass diese Menge endlich ist, denn in [14] wurde folgender Satz bewiesen, wobei eine k-Sterne Reduktion die natürliche Verallgemeinerung einer 4-Sterne Reduktion bezeichnen soll.

(2.31) Satz
Für jede natürliche Zahl $k \geq 4$ gibt es keine endliche TW_k-vollständige Menge von TW_k-sicheren k-Sterne Reduktionen.

3 Reduktionsregeln

In diesem Kapitel werden alle TW_4-sichere Reduktionen vorgestellt, die eine TW_4-vollständigen Menge bilden. Auf diesen wird unser Algorithmus aus Kapitel 4 aufbauen. Alle, bis auf die in Abschnitt 3.1, wurden von Daniel P. Sanders gefunden und können auch in [16] oder [17] nachgelesen werden. Zunächst werden alle Reduktionen definiert, doch erst im letzten Abschnitt wird klar, warum diese eine TW_4-vollständige Menge bilden.

Wir beginnen mit einer Reduktion, die für unseren Algorithmus zwar nicht notwendig ist, die uns aber hilft die Sicherheit anderer Reduktionen zu zeigen.

(3.1) Stern-O Reduktion
Die Reduktion $R_{\mathrm{SO}} = (S_{\mathrm{SO}}, T_{\mathrm{SO}})$ wird definiert durch Abbildung 3.1.

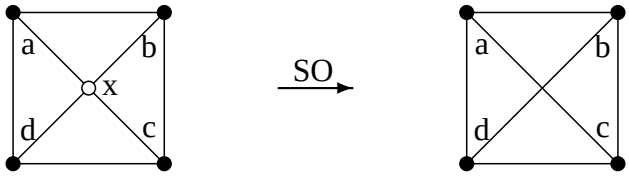

Abbildung 3.1: SO Reduktion

In allen Abbildungen von Strukturen dieses Kapitels sollen schwarze Knoten die Ankerknoten bezeichnen und weiße die inneren Knoten.

(3.2) Lemma
Die SO Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis
Seien G und H Graphen mit $G \leq_{SO} H$. Wir können o.B.d.A. annehmen, dass G aus H entstanden ist durch Anwendung von SO. Falls $G \in TW_4$, dann ist mit Lemma 2.29 auch $H \in TW_4$, da $G = H * x$. Damit folgt vor allem, dass SO eine 4-Sterne Reduktion ist. Falls $H \in TW_4$, dann ist entweder $H + ac \in TW_4$ oder $H + bd \in TW_4$, da (a, b, c, d) ein Kreis der Länge 4 ist und mit Korollar 2.20 eine Sehne hinzugefügt werden kann, ohne die Baumweite zu erhöhen. Da $G \cong (H + ac) \cdot bx \cong (H + bd) \cdot ax$ ist in beiden Fällen mit Lemma 2.28 $G \in TW_4$. $\mathcal{QED}$

Der Beweis zeigt uns auch, wie wir die gewonnenen Ergebnisse aus Kapitel 2, wie Korollar 2.20, Lemma 2.28 und Lemma 2.29, dazu benutzen können um die Sicherheit anderer Reduktionen zu zeigen.

3.1 TW3-sichere Reduktionen

Zunächst wird die TW_3-vollständige Menge von TW_3-sicheren Reduktionen vorgestellt, die als erstes von Arnborg und Proskurowski in [4] gefunden wurden. Wir werden sehen, dass diese auch TW_4-sicher sind.

(3.3) TW2 Reduktionen
Die Reduktionen $R_{\text{Zero}} = (S_{\text{Zero}}, T_{\text{Zero}})$, $R_{\text{One}} = (S_{\text{One}}, T_{\text{One}})$ und $R_{\text{Series}} = (S_{\text{Series}}, T_{\text{Series}})$ werden durch Abbildung 3.2 definiert.

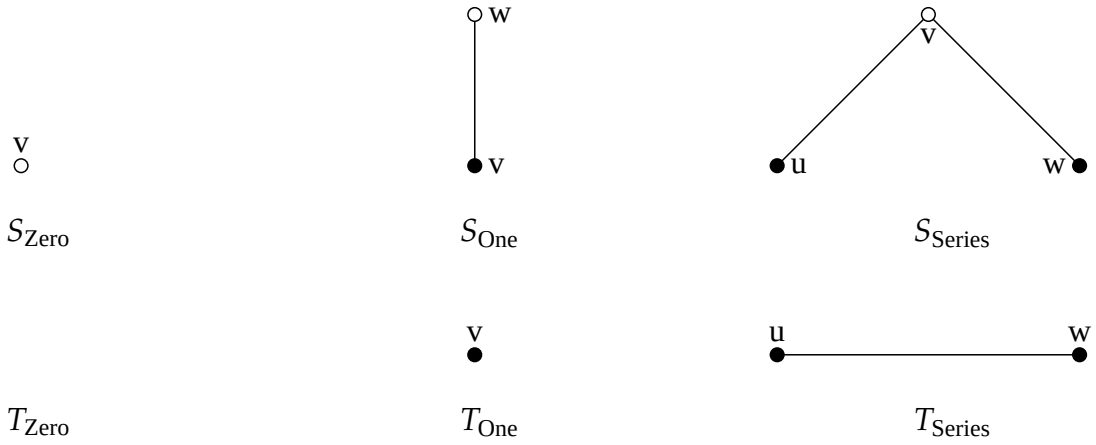

Abbildung 3.2: TW_2-sichere Reduktionen

Durch die ersten beiden Reduktionen werden Graphen mit Baumweite 1, also Bäume und Wälder, beschrieben. Das ist ziemlich offensichtlich. Mit Zuhilfenahme der letzten Reduktionsregel kann man Graphen mit Baumweite 2, also series-parallel Graphen, charakterisieren.

(3.4) TW3 Reduktionen
Die Reduktionen $R_{\text{Triangle}} = (S_{\text{Triangle}}, T_{\text{Triangle}})$, $R_{\text{Buddy}} = (S_{\text{Buddy}}, T_{\text{Buddy}})$ und $R_{\text{Cube}} = (S_{\text{Cube}}, T_{\text{Cube}})$ werden durch Abbildung 3.3 definiert.

Diese 6 Reduktionen bilden eine TW_3-vollständigen Menge von TW_3-sicheren Reduktionen. Wir werden in Abschnitt 3.4 sehen, dass die *Cube*-Reduktion in der Menge CM enthalten ist. Nun sehen wir, dass diese auch TW_4-sicher sind, mithilfe von Minoren und Knoten-Eliminierungen.

(3.5) Lemma
Die Reduktionen aus (3.3) und (3.4) sind TW_4-sichere 4-Sterne Reduktionen.

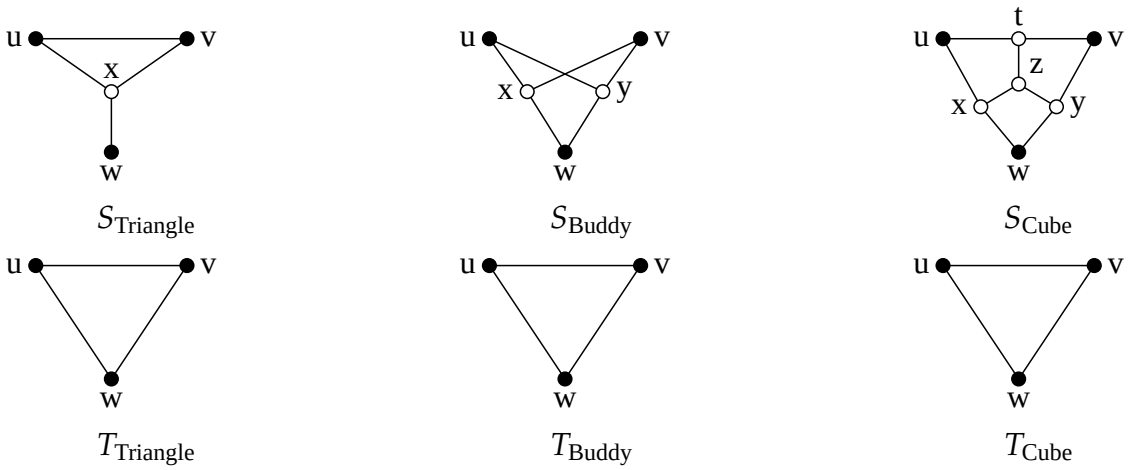

Abbildung 3.3: TW_3-sichere Reduktionen

Beweis
Sei $R = (S_R, T_R)$ eine der genannten Reduktionen. Man sieht, dass für alle Reduktionen T_R aus S_R einerseits durch Knoten-Eliminierungen von Knoten vom Grad höchstens 4 entsteht, andererseits auch durch Kanten-Kontraktionen. Also gilt $T_R \leq_* S_R$ und $T_R \leq_m S_R$ und somit für Graphen G und H mit $G \leq_R H$ auch $G \leq_* H$ und $G \leq_m H$. Mit Lemma 2.30 folgt die Behauptung. $\mathcal{QED}$

3.2 Einfache Y-Δ Reduktionen

Dieser und der nächste Abschnitt handeln von Reduktionen, die durch Knoten-Eliminierungen vom Grad 3 beschrieben werden können. Deswegen nennt man diese Y-Δ Reduktionen. Zunächst betrachten wir einfache Y-Δ Reduktionen, deren Sicherheit ähnlich wie in 3.1 gezeigt werden kann.

(3.6) YO und H7 Reduktionen
Die Reduktionen $R_{\text{YO}} = (S_{\text{YO}}, T_{\text{YO}})$ und $R_{\text{H7}} = (S_{\text{H7}}, T_{\text{H7}})$ werden durch Abbildung 3.4 definiert.

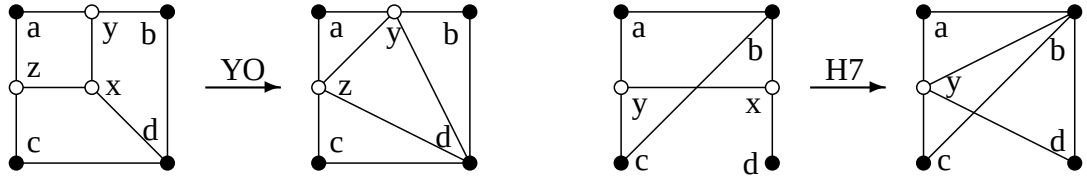

Abbildung 3.4: YO und H7 Reduktionen

(3.7) Lemma
Die YO Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis
Klar ist YO eine 4-Sterne Reduktion, da $G(T_{YO}) \leq_* G(S_{YO})$. Seien G und H gegeben mit $G \leq_{YO} H$, wobei wir o.B.d.A. annehmen können, dass G aus H entstanden ist durch Anwendung von YO. Falls $G \in TW_4$, dann ist mit $G = H * x$ und Lemma 2.29 $H \in TW_4$. Sei nun $H \in TW_4$. Da (a,y,x,z) ein Kreis der Länge 4 ist, können wir mit Korollar 2.20 eine Sehne einfügen, ohne die Baumweite zu erhöhen. Falls $H + ax \in TW_4$, dann enthält $J = (H + ax) \cdot by \cdot cz$ die Struktur S_{SO} bei x und somit ist mit Lemma 2.28 und Lemma 3.2 $J * x \in TW_4$. Wegen $J * x = G * y * z$ ist dann mit Lemma 2.29 $G \in TW_4$. Andernfalls ist $H + yz \in TW_4$ und, da $H + yz$ die Struktur $S_{Triangle}$ bei x besitzt, auch $(H + yz) * x \in TW_4$ mit Lemma 3.5. Also auch $G = H * x = (H + yz) * x$. $\mathcal{QED}$

(3.8) Lemma
Die $H7$ Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis
Wieder ist klar, dass $H7$ eine 4-Sterne Reduktion ist. Seien wieder G und H gegeben, sodass G aus H entstanden ist durch Anwendung von $H7$. Wenn $G \in TW_4$ ist, dann ist mit $G = H * x$ und Lemma 2.29 auch $H \in TW_4$. Falls $H \in TW_4$ ist, dann ist wieder mit Korollar 2.20 entweder $H + ax \in TW_4$ oder $H + by \in TW_4$. Falls $H + ax \in TW_4$ ist, ist auch mit Lemma 2.28 $J = (H + ax) \cdot cy \cdot dx \in TW_4$ und somit wegen Lemma 2.29 und $J = G * y$ auch $G \in TW_4$. Andernfalls ist $H + by \in TW_4$ und da $H + by$ die Struktur $S_{Triangle}$ bei x besitzt, auch $(H + by) * x \in TW_4$ mit Lemma 3.5. Somit auch $G = H * x = (H + by) * x$. $\mathcal{QED}$

(3.9) TO und YI Reduktionen
Die Reduktionen $R_{\mathrm{TO}} = (S_{\mathrm{TO}}, T_{\mathrm{TO}})$ und $R_{\mathrm{YI}} = (S_{\mathrm{YI}}, T_{\mathrm{YI}})$ werden durch Abbildung 3.5 definiert.

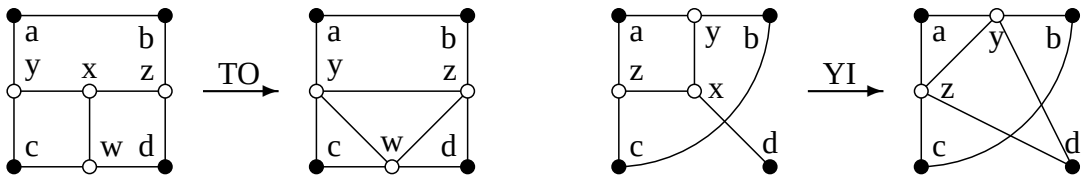

Abbildung 3.5: TO und YI Reduktionen

(3.10) Lemma
Die TO Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis
Klar ist TO eine 4-Sterne Reduktion. Seien G und H gegeben mit $G \leq_{TO} H$, wobei wir o.B.d.A. annehmen können, dass G aus H entstanden ist durch Anwendung von TO. Falls $G \in TW_4$, dann ist wegen $G = H * x$ und Lemma 2.29 $H \in TW_4$. Sei nun $H \in TW_4$. Da (c, y, x, w) ein Kreis der Länge 4 ist, können wir mit Korollar 2.20 eine Sehne einfügen, ohne die Baumweite zu erhöhen. Falls $H + cx \in TW_4$, dann ist $J = (H + cx) \cdot ay \cdot bz \cdot dw \in TW_4$ mit Lemma 2.28. Da J die Struktur S_{SO} bei x besitzt, ist mit Lemma 3.2 $J * x \in TW_4$ und wegen $G * w * y * z = J * x$ und Lemma 2.29 auch $G \in TW_4$. Andernfalls ist $H + yw \in TW_4$, wobei $H + yz$ die Struktur $S_{Triangle}$ bei x besitzt. Also auch $(H + yw) * x \in TW_4$ mit Lemma 3.5 und somit auch $G = H * x = (H + yw) * x$. $\mathcal{QED}$

(3.11) Lemma
Die YI Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis
Wieder ist klar, dass YI eine 4-Sterne Reduktion ist. Seien wieder G und H gegeben, sodass G aus H entstanden ist durch Anwendung von YI. Wenn $G \in TW_4$ ist, dann ist mit $G = H * x$ und Lemma 2.29 auch $H \in TW_4$. Falls $H \in TW_4$ ist, dann ist wieder mit Korollar 2.20 entweder $H + ax \in TW_4$ oder $H + yz \in TW_4$. Falls $H + ax \in TW_4$ ist, ist auch mit Lemma 2.28 $J = (H + ax) \cdot by \cdot cz \cdot dx \in TW_4$ und somit wegen Lemma 2.29 und $J = G * y * z$ auch $G \in TW_4$. Andernfalls ist $H + yz \in TW_4$ und da $H + yz$ die Struktur $S_{Triangle}$ bei x besitzt, auch $(H + yz) * x \in TW_4$ mit Lemma 3.5. Somit auch $G = H * x = (H + yz) * x$. $\mathcal{QED}$

3.3 Y-Δ Leiter Reduktionen

Ziel dieses Abschnittes ist es Leiter Reduktionen zu definieren. Doch um zu zeigen, dass diese TW_4-sicher sind, müssen wir zunächst Dreieck Leiter Reduktionen einführen.

3.3.1 Dreieck Leiter Reduktionen

Mit Dreieck Leiter Reduktionen werden Strukturen eines Graphen durch ähnliche Strukturen ersetzt. Deswegen können diese nicht einfach durch Minoren oder Knoten-Eliminierungen beschrieben werden.

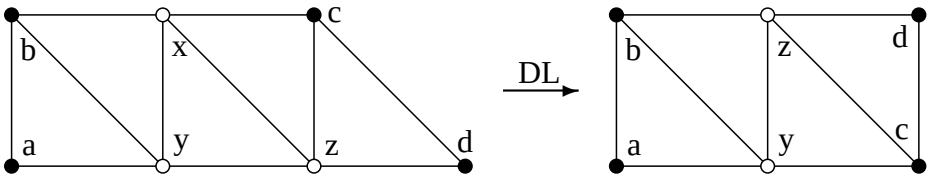

Abbildung 3.6: DL Reduktion

(3.12) Doppel Leiter Reduktion

Die Reduktion $R_{\mathrm{DL}} = (S_{\mathrm{DL}}, T_{\mathrm{DL}})$ wird definiert durch Abbildung 3.6.

(3.13) X Leiter 1 Reduktion

Die Reduktion $R_{\mathrm{XL1}} = (S_{\mathrm{XL1}}, T_{\mathrm{XL1}})$ wird definiert durch Abbildung 3.7.

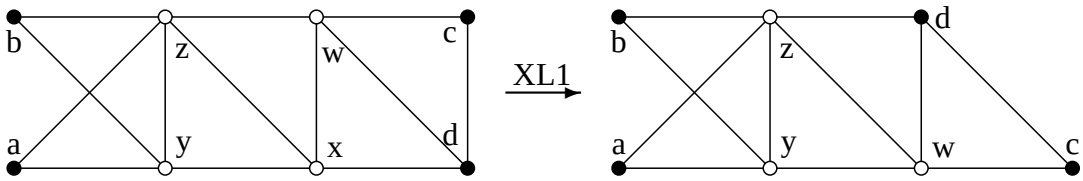

Abbildung 3.7: XL1 Reduktion

(3.14) X Leiter 2 Reduktion

Die Reduktion $R_{\mathrm{XL2}} = (S_{\mathrm{XL2}}, T_{\mathrm{XL2}})$ wird definiert durch Abbildung 3.8.

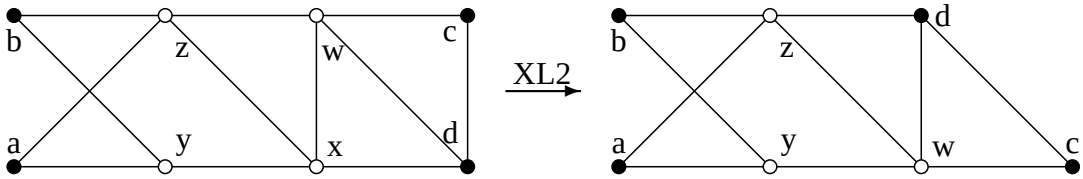

Abbildung 3.8: XL2 Reduktion

(3.15) Stern Leiter Reduktion

Die Reduktion $R_{\mathrm{SL}} = (S_{\mathrm{SL}}, T_{\mathrm{SL}})$ wird definiert durch Abbildung 3.9.

Man beachte, dass es sich hierbei nicht um 4-Sterne Reduktionen handelt, also sind diese für unseren Algorithmus unpraktisch. Wie bereits erwähnt, lassen sich diese Reduktionen nicht mit Minoren oder Knoten-Eliminierungen beschreiben, somit müssen wir auf Baumzerlegungen zurückgreifen um die Sicherheit dieser Reduktionen zu beweisen. Folgende Bezeichnungen sind dabei nützlich.

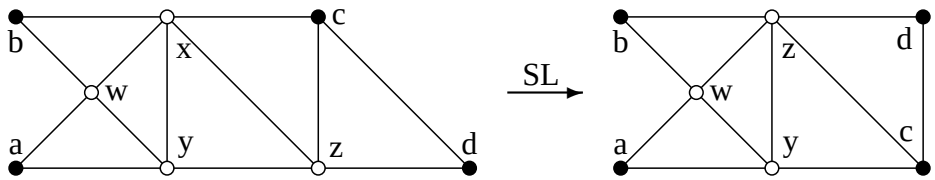

Abbildung 3.9: SL Reduktion

(3.16) Bezeichnungen
Für einen Baum T und verschiedene Knoten $r, s, t \in V(T)$ wird der eindeutige Knoten, der auf dem Pfad von r nach s, auf dem Pfad von r nach t und auf dem Pfad von s nach t liegt, als *Schwerpunkt* bezeichnet. Für eine Baumzerlegung (T, B) eines Graphen G bezeichne mit $G(T, B)$ den Graphen mit Knotenmenge $V(G)$ und Kantenmenge $\{\{x, y\} | \exists t \in V(T) \text{ mit } \{x, y\} \subseteq B_t\}$.

Es ist leicht zu sehen, dass es für je 3 Knoten eines Baumes T genau einen Schwerpunkt gibt. Auch ist leicht zu sehen, dass eine Baumzerlegung (T, B) eines Graphen G selbst wieder eine Baumzerlegung von $G(T, B)$ ist.

(3.17) Lemma
Die Reduktionen DL, $XL1$, $XL2$ und SL sind TW_4-sicher.

Wir wollen den Beweis exemplarisch für die DL Reduktion führen, alle anderen lassen sich auf eine ähnliche Weise beweisen und können in [17] nachgelesen werden.

Beweis
Seien Graphen G und H mit $G \leq_{DL} H$ gegeben, wobei wir o.B.d.A. annehmen können, dass G aus H entstanden ist durch Anwendung von DL. Sei zunächst $G \in TW_4$ und dazu eine passende Baumzerlegung (T, B) von G der Weite 4 gegeben. Wenn es eine Tasche B_t, $t \in V(T)$, gibt, sodass $|B_t \cap \{a, b, c, d, y, z\}| \geq 4$, dann ist $J = G * y * z \leq_m G(T, B)$, da J der vollständige Graph auf 4 Knoten ist und $G(T, B)$ eine 4-Clique enthält. Das sagt uns mit Lemma 2.28, dass $J \in TW_4$ ist. Also ist wegen $H * y * x * z = J$ und Lemma 2.29 auch $H \in TW_4$. Also können wir jetzt annehmen, dass es keine solche Tasche gibt. Wegen Lemma 2.5 wissen wir, dass es Knoten $t_1, t_2, t_3, t_4 \in V(T)$ gibt, mit $\{a, b, y\} \in t_1$, $\{b, y, z\} \in t_2$, $\{c, y, z\} \in t_3$ und $\{c, d, z\} \in t_4$, wobei wir ohne Einschränkung sagen können, dass t_2, t_3, t_4 minimalen Abstand (im Sinne eines kürzesten Weges) zu t_1 haben. Laut unserer Annahme sind alle Knoten t_1, t_2, t_3, t_4 verschieden. Sei c_2 der Schwerpunkt von t_1, t_2, t_3. Man beachte, dass $\{b, y\} \subseteq (B_{t_1} \cap B_{t_2})$ und das c_2 auf dem Pfad von t_1 nach t_2 liegt. Man beachte weiter, dass $z \in (B_{t_2} \cap B_{t_3})$ und c_2 auf dem Pfad von t_2 nach t_3 liegt. Aus der dritten Bedingung der Definition einer Baumzerlegung wissen wir, dass $\{b, y, z\} \subseteq B_{c_2}$. Weil t_2 der Knoten mit minimalem Abstand ist, der diese Bedingung erfüllt, ist $c_2 = t_2$. Analog folgt auch dass t_3 der Schwerpunkt von t_2, t_3, t_4 ist. Somit wissen wir, dass es

einen Pfad P in T gibt, beginnend bei t_1, zunächst über t_2, dann über t_3 und endend bei t_4. Sei nun t der Nachbar von t_3, sodass t_1 und t_3 in verschiedenen Komponenten von $T - tt_3$ liegen. Sei $t_{\text{neu}} \notin V(T)$ ein neuer Knoten und $S = T - tt_3 + t_{\text{neu}} + tt_{\text{neu}} + t_{\text{neu}}t_3$. Weiter definieren wir die Komponenten P_1 von $P - t_2$, die t_1 enthält, P_2 von $(P \setminus P1) - t_3$, die t_2 enthält und P_4 von $P - t_3$, die t_4 enthält. Für jeden Knoten $s \in V(T \setminus P)$ sei $A_s = B_s \setminus \{y,z\}$. Für jeden Knoten $s \in V(P_1)$ sei $A_s = B_s$. Für jeden Knoten $s \in V(P_2)$ sei $A_s = B_s \setminus \{z\} \cup \{x\}$. Für jeden Knoten $s \in V(P_4)$ sei $A_s = B_s \setminus \{y\}$. Weiter sei $A_{t_{\text{neu}}} = B_{t_3} \setminus \{c\} \cup \{x\}$ und $A_{t_3} = B_{t_3} \setminus \{y\} \cup \{x\}$. Jetzt ist leicht zu sehen, dass mit $A = \cup_{s \in V(S)}\{A_s\}$, (S,A) eine Baumzerlegung von H der Weite höchstens 4 ist und somit $H \in TW_4$. Sei jetzt umgekehrt $H \in TW_4$ und $M = H - (V(H) \setminus \{a,b,c,x,y,z\})$. Definiere J als den Graphen, der aus H entstanden ist durch Ersetzen von T_{DL} durch S_{DL}, sodass $(M,(a,b,z,c))$ die Kopie von T_{DL} in H ist. Also ist $H \leq_{DL} J$ und mit dem ersten Teil dieses Beweises $J \in TW_4$. Dann ist auch $G \leq_m J$ und mit Lemma 2.28 $G \in TW_4$. $\mathcal{QED}$

Diese 4 Reduktionen können wir nun im nächsten Unterabschnitt verwenden.

3.3.2 Leiter Reduktionen

Die nächsten 4 Reduktionen sind wieder 4-Sterne Reduktionen und werden in unserem Algorithmus verwendet. Mit Hilfe der Reduktionen aus Abschnitt 3.3.1 ist die Sicherheit wieder leicht zu zeigen.

(3.18) Leiter 1 Reduktion
Die Reduktion $R_{\text{L1}} = (S_{\text{L1}}, T_{\text{L1}})$ wird definiert durch Abbildung 3.10.

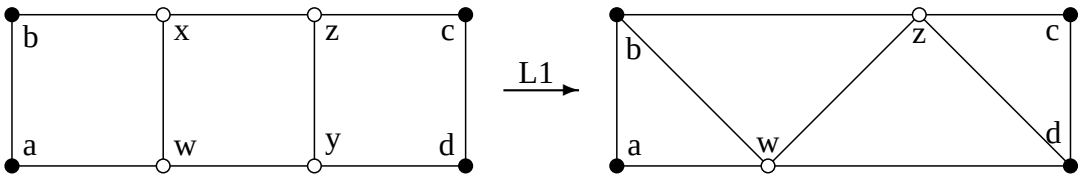

Abbildung 3.10: L1 Reduktion

(3.19) Lemma
Die $L1$ Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis
Klar ist, dass $L1$ eine 4-Sterne Reduktion ist. Seien G und H gegeben, sodass G aus H entstanden ist durch Anwendung von $L1$. Falls $G \in TW_4$ ist, dann ist mit Lemma 2.29 und $G = H * x * y$

auch $H \in TW_4$. Wenn andererseits $H \in TW_4$ ist, können wir mit Korollar 2.20 eine Sehne zu (x,w,y,z) hinzufügen ohne die Baumweite zu erhöhen. Falls $H + wz \in TW_4$, dann mit Lemma 2.28 auch $G = (H + wz) \cdot bx \cdot dy \in TW_4$. Andernfalls ist $H + xy \in TW_4$ und somit auch $J = (H + xy) \cdot aw \cdot cz \in TW_4$, wobei J die Struktur T_{DL} enthält. Also gibt es ein K, sodass $J \leq_{DL} K$ (zweimaliges Anwenden von DL) und $G \leq_m K$. Das zeigt mit Lemma 2.28 und Lemma 3.17, dass $G \in TW_4$ ist. $\mathcal{QED}$

(3.20) Leiter 2 Reduktion

Die Reduktion $R_{\mathrm{L2}} = (S_{\mathrm{L2}}, T_{\mathrm{L2}})$ wird definiert durch Abbildung 3.11.

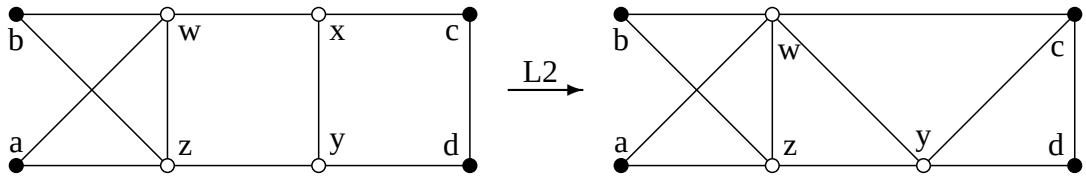

Abbildung 3.11: L2 Reduktion

(3.21) Lemma

Die $L2$ Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis

Klar ist, dass $L2$ eine 4-Sterne Reduktion ist. Seien G und H gegeben, sodass G aus H entstanden ist durch Anwendung von $L2$. Falls $G \in TW_4$ ist, dann ist mit Lemma 2.29 und $G = H * x$ auch $H \in TW_4$. Wenn andererseits $H \in TW_4$ ist, können wir mit Korollar 2.20 eine Sehne zum Kreis (w,z,y,x) hinzufügen ohne die Baumweite zu erhöhen. Falls $H + wy \in TW_4$, definiere $J = (H + wy) \cdot cx$, ansonsten ist $H + zx \in TW_4$ und dann definiere $J = (H + xz) \cdot dy$. In beiden Fällen ist $J \in TW_4$ und es gibt ein K, sodass $J \leq_{XL1} K$ ist (da J die Struktur T_{XL1} enthält) und $G \leq_m K$. Mit Lemma 2.28 und Lemma 3.17 folgt, dass $G \in TW_4$ ist. $\mathcal{QED}$

(3.22) Leiter 3 Reduktion

Die Reduktion $R_{\mathrm{L3}} = (S_{\mathrm{L3}}, T_{\mathrm{L3}})$ wird definiert durch Abbildung 3.12.

(3.23) Lemma

Die $L3$ Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

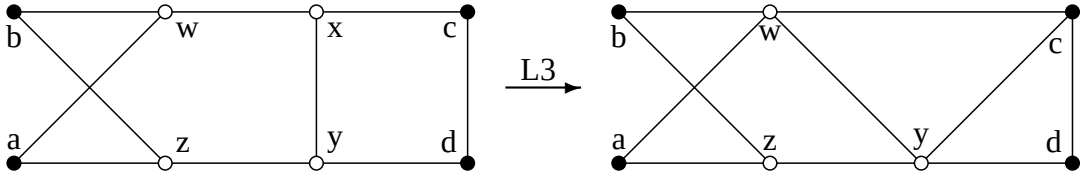

Abbildung 3.12: L3 Reduktion

Beweis

Klar ist, dass $L3$ eine 4-Sterne Reduktion ist. Seien G und H gegeben, sodass G aus H entstanden ist durch Anwendung von $L3$. Falls $G \in TW_4$ ist, dann ist mit Lemma 2.29 und $G = H * x$ auch $H \in TW_4$. Wenn andererseits $H \in TW_4$ ist, können wir mit Korollar 2.20 eine Sehne zum Kreis (b,z,a,w) hinzufügen, ohne die Baumweite zu erhöhen. Falls $H + ba \in TW_4$, dann auch mit Lemma 2.28 $J = (H + ba) \cdot cx \cdot cw \cdot dy \cdot dz$. Wegen $J = G * w * z * y$ und Lemma 2.29 ist auch $G \in TW_4$. Ansonsten ist $H + wz \in TW_4$ und wir können wieder zu (w,z,y,x) eine Sehne hinzufügen ohne die Baumweite zu erhöhen. Falls $H + wz + wy \in TW_4$, definiere $J = (H + wz + wy) \cdot cx$, ansonsten ist $H + wz + zx \in TW_4$ und dann definiere $J = (H + wz + xz) \cdot dy$. In beiden Fällen ist $J \in TW_4$ und es gibt ein K, sodass $J \leq_{XL1} K$ ist (da J die Struktur T_{XL1} enthält) und $G \leq_m K$. Mit Lemma 2.28 und Lemma 3.17 folgt, dass $G \in TW_4$ ist. $\underline{\mathcal{QED}}$

(3.24) Leiter 4 Reduktion

Die Reduktion $R_{\mathrm{L4}} = (S_{\mathrm{L4}}, T_{\mathrm{L4}})$ wird definiert durch Abbildung 3.13.

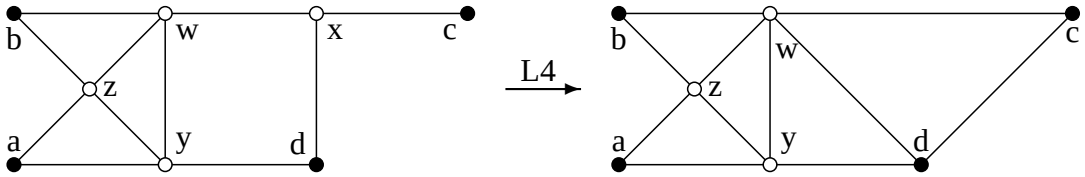

Abbildung 3.13: L4 Reduktion

(3.25) Lemma

Die $L4$ Reduktion ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis

Klar ist, dass $L4$ eine 4-Sterne Reduktion ist. Seien G und H gegeben, sodass G aus H entstanden ist durch Anwendung von $L4$. Falls $G \in TW_4$ ist, dann ist mit Lemma 2.29 und $G = H * x$ auch $H \in TW_4$. Wenn andererseits $H \in TW_4$ ist, können wir mit Korollar 2.20 eine Sehne zu (w,y,d,x) hinzufügen ohne die Baumweite zu erhöhen. Falls $H + wd \in TW_4$, dann mit Lemma 2.28 auch $G = (H + wd) \cdot cx \in TW_4$. Andernfalls ist $H + xy \in TW_4$ und somit auch $J = (H + xy) \cdot cx \in TW_4$, wobei J die Struktur T_{SL} enthält. Also gibt es ein K, sodass $J \leq_{SL} K$ und $G \leq_m K$. Das zeigt mit Lemma 2.28 und Lemma 3.17, dass $G \in TW_4$ ist.

3.4 Superstrukturen

Aus den in den vorherigen Abschnitten gefundenen Reduktionen lässt sich noch nicht schließen, dass diese Graphen mit Baumweite kleiner oder gleich vier charakterisieren. Das wird erst mit folgenden Reduktionen deutlich. Dieser Abschnitt handelt von komplizierteren Strukturen. Diese Strukturen sind Zusammensetzungen aus Blatt Strukturen, welche mögliche Strukturen in der Nähe von Blättern einer Baumzerlegung sind. Diese Strukturen können aufgeteilt werden in 60 einfache Blattstrukturen und 4 unendliche Familien von Blattstrukturen. Wir werden erst diese Blattstrukturen definieren und erst später sehen, wie diese konstruiert wurden und warum Zusammensetzungen aus diesen Strukturen Graphen mit Baumweite kleiner oder gleich vier beschreiben.

(3.26) Definition: Einfache Blatt Strukturen
Die *einfachen Blatt Strukturen* $LS_0, LS_1, \ldots, LS_{59}$ werden durch die Abbildungen 3.14 und 3.15 definiert.

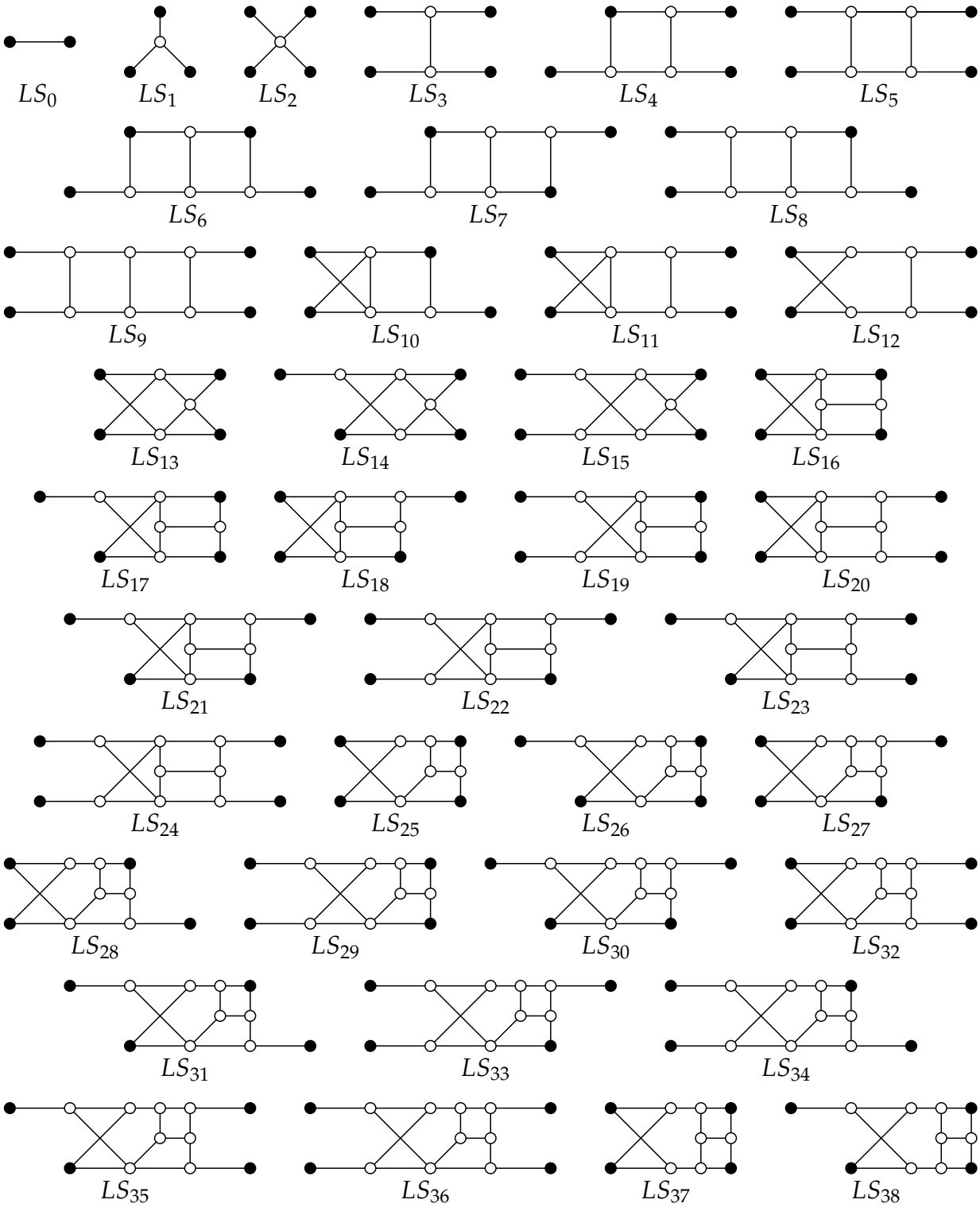

Abbildung 3.14: Einfache Blatt Strukturen ($LS_0, \ldots, LS_{38}$)

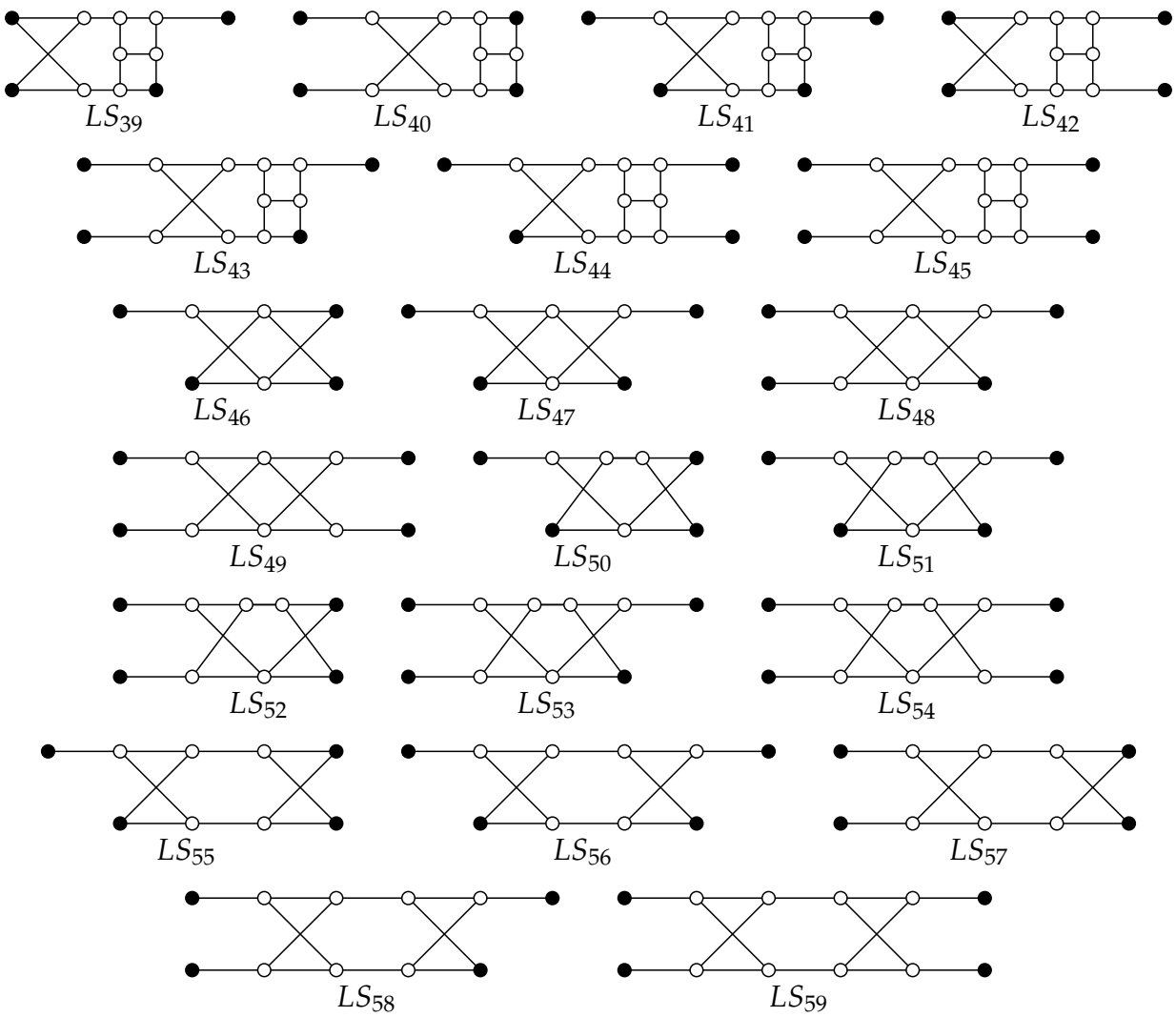

Abbildung 3.15: Einfache Blatt Strukturen ($LS_{39}, \ldots, LS_{59}$)

(3.27) Definition: Unendliche Familien von Blatt Strukturen

Die unendlichen *Familien von Blatt Strukturen* $L_{1,n}, L_{2,n}, L_{3,n}, L_{4,n}$, für $n \in \mathbb{N}^{\geq 1}$, werden durch Abbildung 3.16 definiert. Die Zahl n bestimmt dabei die Anzahl der Knoten: $|V(L_{1,n})| = n + 5$, $|V(L_{2,n})| = n + 5$, $|V(L_{3,n})| = n + 6$ und $|V(L_{4,n})| = n + 6$.

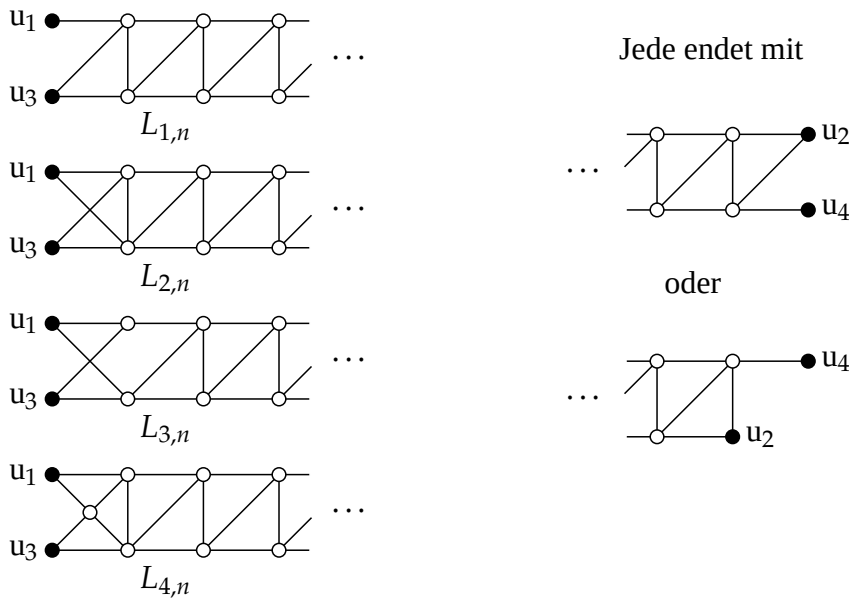

Abbildung 3.16: Familien von Blatt Strukturen

Die folgenden Reduktionen können als Verallgemeinerung der *Buddy* und der *Cube* Reduktion aufgefasst werden. Man könnte annehmen, dass die Blattstruktur LS_1 eine Blattstruktur für TW_3 ist. Dann können beide Reduktionen aus dieser Blattstruktur zusammengesetzt werden, wie Abbildung 3.17 zeigt.

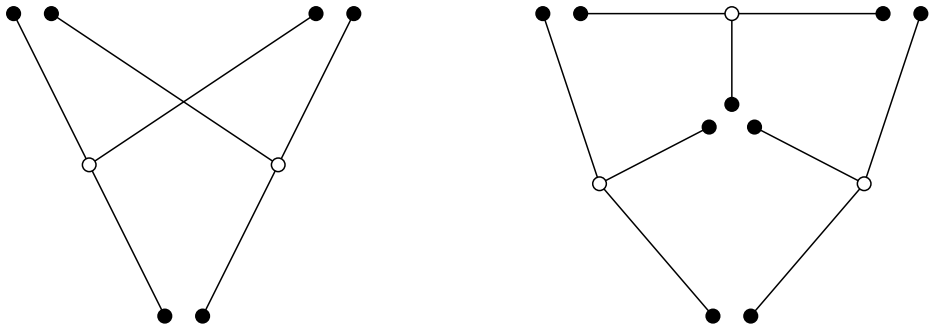

Abbildung 3.17: Die Strukturen S_{Buddy} und S_{Cube} als Zusammensetzungen von LS_1 Strukturen

Mit Hilfe aller Blattstrukturen können wir Superstrukturen definieren, welche Zusammensetzungen aus Blattstrukturen sind.

(3.28) Definition: Superstruktur

Eine *Superstruktur* S ist eine Struktur, die folgendes erfüllt. Die Menge A der Ankerknoten von S besteht aus höchstens 4 Knoten. Es gibt ein *Zentrum* $x \in V(S)$, welcher ein innerer Knoten ist, also $x \notin A$. Der Graph von S ist eine Vereinigung von Graphen einer endlichen Menge L

von Blattstrukturen, sodass für jede Blattstruktur aus L mit der Menge B der Ankerknoten die Bedingung $x \in B \subseteq A \cup \{x\}$ erfüllt ist. Wir sagen, dass S die Blattstrukturen aus L *enthält*.

Eine Superstruktur zeichnet sich also dadurch aus, dass es einen Knoten, das Zentrum, gibt, mit dem jede Blattstrukturen verbunden ist. Zur Veranschaulichung diskutieren wir einige Beispiele.

(3.29) Beispiel
Wir sehen, dass die Struktur S_{Cube} eine Superstruktur ist, die aus drei LS_1 Blattstrukturen besteht, wie Abbildung 3.17 zeigt. Die Abbildung 3.18 zeigt zwei weitere Superstrukturen. Das Zentrum wird dabei grau markiert.

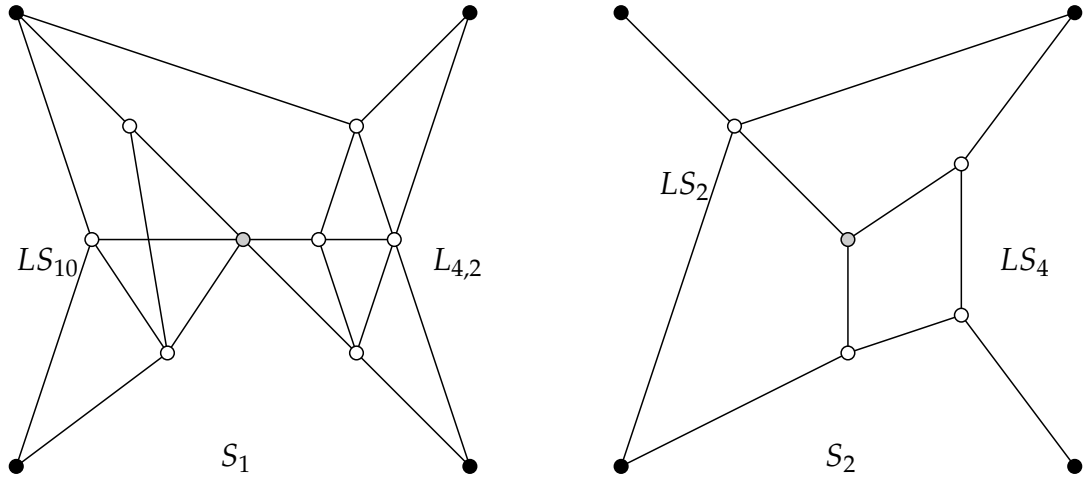

Abbildung 3.18: Zwei Superstrukturen S_1 und S_2

Wir wollen jetzt Superstrukturen auf den vollständigen Graphen auf den Ankerknoten Reduzieren. Da das eine unendliche Menge von Reduktionen ist, brauchen wir eine einfache Methode um die Sicherheit zu zeigen. Deshalb schränken wir diese Menge ein.

(3.30) CM Reduktionen
Die Menge CM sei die Menge aller Reduktionen R, wobei S_R eine Superstruktur ist, $G(T_R)$ der vollständige Graph auf der Menge der Ankerknoten ist und entweder ist $G(T_R) \leq_m G(S_R)$ oder es gibt einen Graphen H, sodass $G(T_R) \leq_{SO} H \leq_m G(S_R)$.

Wenn wir uns die Strukturen S_1 und S_2 aus Beispiel 3.29 noch einmal anschauen, dann sehen wir, dass S_1 den vollständigen Graphen auf vier Knoten als Minor hat. Somit gibt es eine Reduktion aus $R \in CM$, mit $S_R \cong S_1$. Für die Struktur S_2 gilt diese Behauptung nicht, jedoch besitzt diese Struktur eine YO Reduktion.

Wir unterteilen die Reduktionen aus CM, damit wir folgendes Problem vermeiden können. Das Zentrum einer Reduktion aus CM hat einen unbeschränkten Grad. Abhilfe schaffen folgenden Reduktionen, die man als Generalisierung der $Buddy$ Reduktion auffassen kann.

(3.31) Triple Reduktionen
Die Menge Triple besteht aus allen Reduktionen R, mit Ankerknoten (a,b,c,d), $G(T_R)$ ist der vollständige Graph auf der Knotenmenge$\{a,b,c,d\}$ und $G(S_R)$ ist die Vereinigung von genau drei Blatt Strukturen, wobei jede davon eine Teilmenge von $\{a,b,c,d\}$ als Ankerknoten besitzt, keine davon LS_0 ist und keine zwei davon LS_1 sind mit den gleichen drei Ankerknoten.

(3.32) BCM Reduktionen
Die Menge BCM ist die Menge aller Reduktionen R, mit $R \in CM$ und $G(S_R)$ enthält keine Buddy oder Triple Reduktion.

Der Grund warum man die Menge CM in die Mengen BCM, $Triple$ und $\{Buddy\}$ zerlegt ist folgender.

(3.33) Lemma
Das Zentrum jeder Reduktion aus BCM hat Grad höchstens 20.

Beweis
Sei R eine Reduktion aus BCM mit Zentrum x. Da R keine $Triple$ Reduktion enthalten darf kann R höchstens aus acht Blattstrukturen mit vier Ankerknoten bestehen. Man beachte, dass der Grad eines Ankerknoten einer Blattstruktur höchstens zwei ist. Falls R keine Blattstruktur LS_1 enthält, dann hat x höchstens 16 Kanten zu den inneren Knoten der Blattstrukturen. Es können auch Kanten zwischen x und den Ankerknoten von R vorliegen, womit man insgesamt auf 20 kommt. Falls R ein LS_1 enthält, gibt es höchstens sechs Blattstrukturen mit vier Ankerknoten. Daraus folgt, dass x Grad kleiner als 20 hat. Ebenso folgt die Behauptung, wenn R mehr als eine Blattstruktur LS_1 enthält. $\mathcal{QED}$

Nun wird aber die Sicherheit dieser Reduktionen gezeigt.

(3.34) Lemma
Jede Reduktion aus $CM \cup Triple$ ist eine TW_4-sichere 4-Sterne Reduktion.

Beweis
Sei $R \in CM \cup$ Triple gegeben. Weiter seien G und H Graphen mit $G \leq_R H$. Die Blatt Strukturen sind so aufgebaut, dass $G \leq_* H$ leicht zu sehen ist. Denn man kann zuerst die

inneren Knoten der Blattstrukturen eliminieren und anschließend das Zentrum. Daraus folgt vor allem, dass R eine 4-Sterne Reduktion ist. Wenn $R \in CM$, dann gilt nach Definition $G \leq_m H$ oder $G \leq_{SO} J \leq_m H$ für ein geeignetes J. In beiden Fällen folgt die Behauptung mit Lemma 2.30 und Lemma 3.2. Wir zeigen zuerst, dass $G \leq_m H$ für $R \in Triple$. Es ist leicht zu sehen, dass $G(LS_2) \leq_m G(LS_i)$ ist, für $3 \leq i \leq 59$, bzw dass $G(LS_2) \leq_m G(L_{i,n})$, für $1 \leq i \leq 4$ und $n \in \mathbb{N}^{\geq 1}$. Also können wir o.B.d.A. annehmen, dass R nur die Blattstrukturen LS_1 und LS_2 enthält. Dann ist aber leicht zu sehen, dass $G \leq_m H$ ist. Die Behauptung folgt mit Lemma 2.30. $\mathcal{QED}$

Damit haben wir alle Reduktionen zusammen, mit denen wir Graphen mit Baumweite kleiner oder gleich vier charakterisieren können.

(3.35) Definition
Die Menge $CS4$ von Reduktionen wird definiert durch $CS_4 = CS_2 \cup EZ \cup CM \cup \{\text{Buddy}\}$, wobei $CS_2 = \{\text{Zero}, \text{One}, \text{Series}\}$ und $EZ = \{\text{Triangle}, YO, H7, TO, YI, L1, L2, L3, L4\}$ ist.

(3.36) Satz
Die Menge CS_4 ist eine TW_4-vollständige Menge von TW_4-sicheren 4-Sterne Reduktionen.

Beweis
Mit den Lemmata 3.5, 3.7, 3.8, 3.10, 3.11, 3.19, 3.21, 3.23, 3.25 und 3.34 folgt, dass jede dieser Reduktionen eine TW_4-sichere 4-Sterne Reduktion ist. Die Vollständigkeit wird in [16] gezeigt. $\mathcal{QED}$

Dieser Satz wurde bewiesen, indem gezeigt wurde, dass jede Superstruktur entweder eine Blattstruktur ist, oder eine Reduktion in CS_4 enthält. Dazu wurden Superstrukturen genau analysiert. Die Ergebnisse aus [16] sind in Tabelle 3.1 zusammengefasst, dabei ist die Tabelle folgendermaßen aufgebaut. Jede Superstruktur, die keine Reduktion in CS_4 enthält und eine Blattstruktur aus der linken Spalte der Tabelle enthält, ist isomorph zu einer Blattstruktur aus der rechten Spalte. Diese Tabelle wird uns bei der Implementierung des Algorithmus in Kapitel 4 nützlich sein.

Superstruktur enthält	**ist isomorph zu**
Nur Strukturen aus $\{LS_0, LS_1, LS_2, LS_3\}$	$LS_1, LS_2, LS_3, LS_4, LS_6, LS_7, LS_{10}, LS_{12}, LS_{13}, LS_{16}, LS_{25}, LS_{37}, LS_{46}, LS_{50}, LS_{55}, L_{1,1}, L_{2,1}$ oder $L_{3,1}$
LS_{13}, LS_{16} oder LS_{46}	$LS_{14}, LS_{17}, LS_{18}$ oder LS_{47}
$LS_{14}, LS_{17}, LS_{18}$ oder LS_{47}	$LS_{15}, LS_{19}, LS_{20}, LS_{21}$ oder LS_{48}
$LS_{15}, LS_{19}, LS_{20}, LS_{21}$ oder LS_{48}	LS_{22}, LS_{23} oder LS_{49}
LS_{22}, LS_{23} oder LS_{49}	LS_{24}
$(n \geq 1)$ LS_6, LS_{10} oder $L_{3,n}$	LS_8, LS_{11} oder $L_{3,n+1}$
LS_8 oder LS_{11}	LS_9
$L_{2,1}$ oder $L_{1,1}$	$LS_{10}, L_{1,2}, L_{2,2}$ oder $L_{4,1}$
$(n \geq 1)$ $L_{1,2}, L_{2,n+1}$ oder $L_{4,n}$	$L_{1,3}, L_{2,n+2}$ oder $L_{4,n+1}$
$(n \geq 3)$ $L_{1,n}$	$L_{1,n+1}$
LS_4	LS_5, LS_6, LS_7 oder LS_8
LS_5 oder LS_7	LS_8
$LS_{25}, LS_{37}, LS_{50}$ oder LS_{55}	$LS_{26}, LS_{27}, LS_{28}, LS_{38}, LS_{39}, LS_{51}, LS_{52}, LS_{56}$ oder LS_{57}
$LS_{26}, LS_{27}, LS_{28}, LS_{38}, LS_{39}, LS_{51}, LS_{52}, LS_{56}$ oder LS_{57}	$LS_{29}, LS_{30}, LS_{31}, LS_{32}, LS_{40}, LS_{41}, LS_{42}, LS_{53}$ oder LS_{58}
$LS_{29}, LS_{30}, LS_{31}, LS_{32}, LS_{40}, LS_{41}, LS_{42}, LS_{53}$ oder LS_{58}	$LS_{33}, LS_{34}, LS_{35}, LS_{43}, LS_{44}, LS_{54}$ oder LS_{59}
$LS_{33}, LS_{34}, LS_{35}, LS_{43}, LS_{44}, LS_{54}$ oder LS_{59}	LS_{36} oder LS_{45}
$LS_9, LS_{12}, LS_{24}, LS_{36}$ oder LS_{45}	keiner

Tabelle 3.1: Superstrukturen isomorph zu Blattstrukturen

Die Menge der Blattstrukturen wurden wie folgt konstruiert. Bei dieser Konstruktion wurden auch die Reduktionen aus EZ gefunden. Man beginnt mit der Menge $L = \{LS_0\}$. Dann bildet man aus allen Blattstrukturen in L Superstrukturen. Falls diese Superstruktur keine Reduktion in CS_4 enthält, fügt man die Superstruktur zu L hinzu. Dies wurde so lang gemacht, bis jede Superstruktur entweder eine Reduktion in CS_4 enthält oder selbst wieder eine Blattstruktur ist. Das ist auch der Grund, warum es die unendlichen Familien von Blattstrukturen gibt. Da diese sich fortsetzen lassen, ohne eine Reduktion in CS_4 zu enthalten. Abbildung 3.19 zeigt einige Bespiele wie Blattstrukturen aus kleinerer Blattstrukturen zusammengesetzt sind.

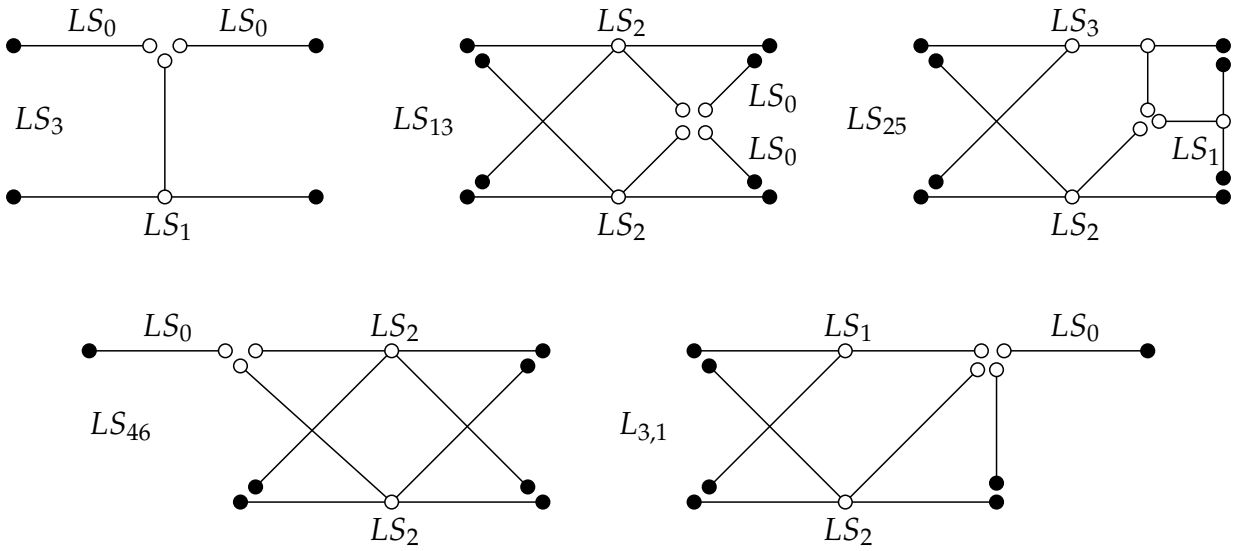

Abbildung 3.19: Superstrukturen isomorph zu Blattstrukturen

Der Beweis der Vollständigkeit wird nun skizziert. Für den formellen Beweis wären noch einige Definitionen von Nöten, welche für unser Vorhaben irrelevant sind. Deshalb vereinfachen wir die Sache, indem wir für einen Graphen $G \in TW_4$ annehmen, dass dieser eine Baumzerlegung (T, B) der Weite vier besitzt, die folgende Bedingungen erfüllt. Zu jedem Knoten (bis auf eine mögliche Ausnahme, diese nennen wir die Wurzel) $t \in V(T)$ gibt es genau einen Nachbarn $s \in V(T)$, welcher $B_t \setminus B_s = \{x\}$ erfüllt, sodass für jeden weiteren Nachbarn $u \in V(T)$ gilt $x \in B_u$. Die Wurzel w muss die Bedingung erfüllen, dass ein Knoten $x \in B_w$ existiert, der in B_u enthalten ist, für jeden Nachbarn u von w. Der Knoten x wird in beiden Fällen das Zentrum einer Superstruktur sein. Mit etwas Übung ist zu sehen, dass eine solche Baumzerlegung durch Modifikation der im Beweis von Satz 2.10 konstruierten Baumzerlegung geschaffen werden kann.

In den Blättern gibt es also einen Knoten, der nur zu den Knoten aus der Tasche in der er enthalten ist adjazent ist. Somit hat dieser Knoten einen Grad kleiner oder gleich vier. Falls der Grad kleiner oder gleich zwei ist, liegt eine der Reduktionen *Zero*, *One* oder *Series* vor. Ansonsten liegt dieser Knoten in einer der Blattstrukturen LS_1 oder LS_2. Falls an keinem Blatt eine Reduktion vorliegt, dann wenden man sich der nächsten Ebene zu, nämlich den Nachbarn der Blätter. Diese haben, da die Baumzerlegung geeignet gewählt wurde, einen Knoten, der in jedem Blatt enthalten ist, mit denen er adjazent ist. Weiter ist dieser Knoten nicht in der Tasche des Nachbarn, welcher kein Blatt ist. Somit bildet diese Tasche und die Blätter, die damit adjazent sind eine Superstruktur. Somit gilt, dass diese entweder eine Reduktion in CS_4 enthält, oder selbst wieder eine Blattstruktur ist. Falls sie eine Blattstruktur ist, wiederholt man diese Schritte. Am Ende erhält man, dass G entweder eine Reduktion in CS_4 enthält oder selbst eine Blattstruktur ist. Falls letzteres zutrifft, dann haben die Ankerknoten der Blattstruktur keine weiteren Nachbarn. Da alle

höchstens den Grad zwei besitzen, liegt dort aber wieder eine der Reduktionen *Zero*, *One* oder *Series* vor.

(3.37) Bemerkung

Man beachte, dass für hinreichend große n die Blattstrukturen $L_{i,n}$ die Reduktion DL besitzen. Man kann also durch Hinzunahme der DL Reduktion zur Menge CS_4 eine endliche TW_4-vollständige Menge von TW_4-sicheren Reduktionen erhalten, da dadurch die Menge der Blattstrukturen und somit auch die Menge der CM Reduktionen endlich ist. Jedoch verzichtet man damit auf 4-Sterne Reduktionen, was die Konstruktion einer 4-Eliminationsfolge erschwert.

4 Algorithmus zur Findung von 4-Eliminationsfolgen

In diesem Kapitel wird ein Algorithmus vorgestellt, der in linearer Zeit feststellt ob die Baumweite eines gegebenen Graphen kleiner oder gleich vier ist, oder ob diese größer als vier ist. Der Algorithmus baut auf den in Kapitel 3 gefundenen Reduktionen auf. Der Ablauf des Algorithmus ist wie folgt. Zunächst wird im Graphen G eine Reduktion aus CS_4 gesucht und angewendet. Somit erhält man einen kleineren Graphen H. Dann wendet man diesen Schritt auf den Graphen H an u.s.w. Da die Reduktionen aus CS_4 alle TW_4 sicher sind, ist $G \in TW_4$ genau dann, wenn $H \in TW_4$ ist. Falls $G \in TW_4$ ist, erhält man am Ende den leeren Graphen, da in jedem Schritt der Graph Baumweite kleiner oder gleich vier hat und die Menge CS_4 TW_4-vollständig ist. Ansonsten terminiert der Algorithmus mit einem nicht-leeren Graphen, der keine Reduktion aus CS_4 besitzt.

Man beachte, dass der Algorithmus nicht nur feststellt, ob ein Graph in TW_4 ist oder nicht, sondern die Mitgliedschaft in TW_4 durch eine 4-Eliminationsfolge beweist. Da alle Reduktionen aus CS_4 4-Sterne Reduktionen sind, ist diese Eliminationsfolge leicht zu konstruieren, und zwar indem man beim Anwenden von Reduktionen die Reihenfolge der eliminierten Knoten ausgibt.

4.1 Linearer Algorithmus

Die Anwendung einer Reduktion bzw. einer Knoten-Eliminierung erfordert das Hinzufügen von Kanten. Damit der Algorithmus linear ist, muss der Zeitaufwand hierfür konstant sein. Da man in einem einfachen Graphen prüfen muss, ob die Kante schon vorhanden ist, werden die Graphen im Algorithmus Multigraphen sein, da sie die Bedingung erfüllen. Der Algorithmus wird somit die Reduktionen auf den zugrunde liegenden einfachen Graphen anwenden. Dafür müssen die Mehrfachkanten an bestimmten Stellen wieder entfernt werden. Dafür ist das folgende Lemma hilfreich.

(4.1) Lemma
Zu einem Multigraphen G und einem Knoten $x \in V(G)$ existiert ein Algorithmus, welcher feststellt, ob x Grad kleiner oder gleich k im zugrunde liegenden einfachen Graphen von G besitzt, eine Anzahl m von Mehrfachkanten inzident mit x entfernt und eine Laufzeit von $O(k^2 + km)$ besitzt.

Dieser Algorithmus wird in Abschnitt 4.2 als Hilfsfunktion vorgestellt. Es wird sich zeigen, dass die Laufzeit dieser Funktion über den ganzen Algorithmus linear ist.

In der Definition 2.23 wurde die Einschränkung gemacht, dass wenn ein Graph eine Reduktion R besitzt, jede Kante zwischen zwei Ankerknoten von R mit einem Knoten inzident ist, welcher Grad kleiner oder gleich sechs hat. Auch das ist notwendig für den Algorithmus, da ansonsten nicht in konstanter Zeit überprüft werden kann, ob zwei Ankerknoten adjazent sind. Somit werden wir das Lemma 4.1 nur für $k \leq 6$ gebrauchen.

Es ist in konstanter Zeit leicht festzustellen, ob ein Knoten x ein Knoten einer Reduktion $R \in CS_2$ (hier muss einfach die Bedingung $deg(x) \leq 2$ geprüft werden) oder einer Reduktion $R \in EZ$ ist, da hierfür nur eine kleiner Umgebung von x betrachtet werden muss. Es wird komplizierter festzustellen, ob x ein Knoten einer $Buddy$ oder einer Reduktion aus CM ist. Bei der $Buddy$ Reduktion kann man nicht einfach die Umgebung eines Knoten betrachten, da die inneren Knoten nicht adjazent sind und die Ankerknoten keinen beschränkten Grad besitzen, wäre der Aufwand mit dieser Methode nicht konstant.

Aus diesem Grund muss der Algorithmus die Blattstrukturen, die x als inneren Knoten enthalten, speichern und ständig aktualisieren. Dabei reicht es aus, die $L_{i,n}$s für die größtmöglichen n zu speichern. Somit kann ein Knoten nur in einer beschränkten Anzahl von Blattstrukturen als innerer Knoten auftreten. Da man für die einfachen Blattstrukturen $LS_1, \ldots, LS_{59}$ einfach wieder die Umgebung des Knoten x betrachten muss, ist dies wieder in konstanter Zeit möglich. Die Blattstrukturen $L_{1,n} \ldots L_{4,n}$ müssen auf eine andere Weise bestimmt werden, da der Aufwand hierfür im schlechtesten Fall linear ist. Zunächst wird geprüft ob x bereits in einer Blattstruktur $L_{i,n}$ enthalten ist. Wenn dies der Fall ist, wird nur an den Ankerknoten geprüft ob x Teil einer größeren Blattstruktur $L_{i,m}$ mit $m > n$ ist. Da gleichzeitig die $L_{i,n}$ für alle inneren Knoten der Blattstrukturen bestimmt werden, kann der Aufwand aufgeteilt werden und ist für jeden Knoten konstant.

Die folgenden Algorithmen wurden aus [16] lediglich übersetzt. Bei der Umsetzung wurde an einer Stelle von den Algorithmen abgewichen, dies wird aber in Abschnitt 4.2 erläutert. Die Algorithmen werden im folgenden vereinfacht erklärt.

Algorithmus 4.1 TREE_WIDTH_FOUR?(G)

if $|E(G)| > 4|V(G)|$ **then**
  Ausgabe: $G \notin TW_4$
end if

$H := G$
Initialisiere VS als leeren Stapel

for $v \in V(G)$ **do**
  lege v auf VS
  $L(v) := \emptyset$
end for

for $e \in E(G)$ **do**
  markiere e
end for

repeat
  nimm einen Knoten x aus VS
  if $deg(x) \leq 2$ **then**
    entferne alle Blattstrukturen, die x enthalten, aus $L(v)$, ersetze dabei $L_{i,n}s$ durch die zur zeit größten bekannten $L_{i,m}s$
    wende die Reduktion aus CS_2 auf H an
    lege die Ankerknoten auf VS
  else if $deg(x) \leq 6$ **then**
    if x ist ein Knoten einer Reduktion $R \in EZ$ **then**
      entferne alle Blattstrukturen, die innere Knoten enthalten, aus $L(v)$, ersetze dabei $L_{i,n}s$ durch die zur zeit größten bekannten $L_{i,m}s$
      wende die Reduktion R auf H an
      lege die Knoten von T_R auf VS
    else
      bestimme die Blattstrukturen $LS_1, \ldots LS_{59}$, die x als inneren Knoten enthalten
      bestimme die größten Blattstrukturen $L_{1,n}, \ldots L_{4,n}$, die x als inneren Knoten enthalten
      for L ist eine neu gefundene Blattstruktur **do**
        for $a \in N(L)$ **do**
          markiere die Kanten von a in L
          CenterCheck(a,$N(L) \setminus \{a\}$,$\{L\}$)
          setze die Kanten von a in L als nicht markiert
        end for
      end for
    end if
  else
    CenterCheck(x,$\emptyset$,$\emptyset$)
  end if

until VS ist nicht leer

if $H = K_0$ **then**
$G \in TW_4$
gebe die Knoten in der Reihenfolge aus, in der sie eliminiert wurden
else
$G \notin TW_4$
end if

Algorithmus 4.2 CenterCheck(a,A,S)

if $|A| > 4$ **then**
return false
end if
if jede Kante inzident mit a ist markiert **then**
if $\bigcup S$ ist eine Blattstruktur **then**
setze jede Kante mit a als nicht markiert
return false
else
entferne alle Blattstrukturen, die innere Knoten enthalten, aus $L(v)$, ersetze dabei $L_{i,n}s$ durch die zur zeit größten bekannten $L_{i,m}s$
wende die Reduktion aus BCM an
lege die Ankerknoten auf VS
return true
end if
end if
repeat
sei α nächste nicht markierte Kante inzident mit $b \neq a$
if eine Kante von a nach b wurde vorher markiert **then**
lösche die Mehrfachkante
end if
until α ist keine Mehrfachkante
markiere α
sei K_1 der Graph mit Knoten $\{a, b\}$ und Kante $\{\{a, b\}\}$
CenterCheck(a,$A \cup \{b\}$,$S \cup \{K_0\}$);
setze α als nicht-markiert
for $L \in L(b)$ mit $a \in N(L)$ **do**
if L ist isomorph zu LS_1 und es gibt ein $M \in S$ mit $N(M) = N(L)$ **then**

entferne alle Blattstrukturen, die a oder b enthalten, aus $L(v)$, ersetze dabei $L_{i,n}s$ durch die zur zeit größten bekannten $L_{i,m}s$
wende die $Buddy$ auf H an
lege $N(L)$ auf VS
return true
end if
for jedes Paar P, Q von nicht trivialen Blattstrukturen in S **do**
if $|N(L) \cup N(P) \cup N(Q)| = 4$ **then**
entferne alle Blattstrukturen, die innere Knoten enthalten, aus $L(v)$, ersetze dabei $L_{i,n}s$ durch die zur zeit größten bekannten $L_{i,m}s$
wende die Reduktion aus $Triple$ auf H an
lege $N(L) \cup N(P) \cup N(Q)$ auf VS
return true
end if
end for
markiere die Kanten von $E(L)$ inzident mit a
CenterCheck(a,$A \cup N(L) \setminus \{a\}$, $S \cup \{L\}$)
setze die Kanten von $E(L)$ inzident mit a als nicht markiert
end for
return false

Der Stapel VS enthält Knoten, an denen eine mögliche Reduktion vorliegen kann. Zunächst wird jeder Knoten darauf gelegt. In der Hauptschleife wird ein Knoten x vom Stapel genommen. Danach wird überprüft ob x ein Knoten einer Reduktion aus $CS_2 \cup EZ$ ist, wenn ja dann wird diese angewendet. Ansonsten werden die Blattstrukturen gesucht, bei denen x ein innerer Knoten ist und diese in die Menge $L(x)$ eingefügt. Für jede neue Blattstruktur wird überprüft ob sie in einer Superstruktur enthalten ist. Dies übernimmt die Funktion CenterCheck. Zum Schluss wird noch überprüft ob x das Zentrum einer Superstruktur ist, ebenfalls mit CenterCheck. Jedes mal, wenn eine Reduktion ausgeführt wird, werden die betroffenen Knoten auf den Stapel gelegt. Gleichzeitig werden die Blattstrukturen, die entfernte Knoten enthalten ebenfalls gelöscht. Zum Schluss wird überprüft, ob der Graph noch Knoten besitzt, und eine entsprechende Ausgabe getätigt.

Die Funktion CenterCheck findet eine $Buddy$, eine $Triple$ oder eine BCM Reduktion, oder stellt fest, dass keine Reduktion aus CM vorliegen kann. Die Parameter der Funktion sind wie folgt. a ist das Zentrum, A eine Liste von Ankerknoten und S eine Liste von Blattstrukturen. Bei einem return false wird nur die aktuelle Ebene der Rekursion verlassen. Return true bewirkt, dass alle Rekursionen verlassen werden. Für jede mit a inzidente Kante α wird die Funktion rekursiv

aufgerufen, wobei S um eine Blattstruktur, die α enthält, erweitert wird. Ebenso wird A um die Ankerknoten der Blattstruktur erweitert. Falls zwei Blattstrukturen LS_1 mit den selben Ankerknoten gefunden wurden, hat man eine $Buddy$ Reduktion gefunden. Falls drei Blattstrukturen mit den selben vier verschiedenen Ankerknoten gefunden wurden, hat man eine $Triple$ Reduktion gefunden. Der rekursive Aufruf endet, falls die Kardinalität von A zu groß ist und somit keine Superstruktur mehr vorliegen kann, oder wenn jede mit a inzidente Kante betrachtet wurde. In diesem Fall wird überprüft, ob es sich bei der Superstruktur um eine Reduktion aus BCM handelt, oder um eine Blattstruktur.

(4.2) Satz
Der Algorithmus TREE_WIDTH_FOUR? findet zu einem Graphen G eine 4-Eliminationsfolge oder entscheidet korrekt, dass $G \notin TW_4$.

Beweis
Sei $G = (V, E)$ ein Graph mit n Knoten. Falls die erste if Bedingung erfüllt ist, wird wegen Lemma 2.17 die korrekte Ausgabe gemacht.

Man beachte, dass der Algorithmus immer prüft, ob eine Kante eine Mehrfachkante ist. Wenn dies der Fall ist wird sie umgehend entfernt. Somit stellt die Benutzung von Multigraphen kein Problem dar.

Als nächstes beachte man, dass der Algorithmus terminiert. In jedem Schritt der Hauptschleife wird ein Knoten aus dem Stapel VS entfernt. Zu Beginn wird jeder Knoten auf den Stapel gelegt. Danach wird nur nach der Anwendung einer Reduktion Knoten auf den Stapel gelegt (höchsten 7 Knoten). Da dabei auch ein Knoten aus H entfernt wird, können höchstens n Reduktionen angewendet werden. Also ist die Anzahl aller Knoten die auf VS gelegt werden beschränkt und es kann keine Endlosschleife auftreten. Auch in CENTERCHECK kann keine Endlosschleife auftreten, da bei jedem Aufruf mindestens eine Kante markiert wird und ein Knoten nur mit einer endlichen Anzahl an Kanten inzident sein kann.

Falls $G \notin TW_4$. Da alle Reduktionen, die angewendet werden in CS_4 sind und somit nach Satz 3.36 TW_4-sicher sind, kann der Algorithmus nicht mit $H = K_0$ terminieren.

Falls $G \in TW_4$. Da alle Reduktionen, die angewendet werden in CS_4 sind und somit nach Satz 3.36 TW_4-sicher sind, hat der zugrunde liegende einfache Graph von H in jedem Stadium des Algorithmus die Baumweite kleiner oder gleich vier. Da CS_4 nach Satz 3.36 auch TW_4-vollständig ist, besitzt der zugrunde liegende einfache Graph von H eine Reduktion in CS_4. Es muss gezeigt werden, dass der Algorithmus so eine Reduktion für ein festes H finden kann.

Sei R eine solche Reduktion. Wenn G diese Reduktion bereits enthält, wird sie gefunden, da jeder Knoten von S_R zu beginn auf den Stapel gelegt wird. Ansonsten wurde die Reduktion zu

einem Zeitpunkt vom Algorithmus erstellt. Der einzige Zeitpunkt an dem das passieren kann, ist nach der Anwendung einer anderen Reduktion R'. Es gibt zwei Möglichkeiten weshalb die Reduktion R erstellt wurde. Entweder weil sich der Grad eines Knoten verringert hat oder weil eine Kante hinzugefügt wurde. In beiden Fällen werden die betroffenen Knoten auf den Stapel gelegt. Man beachte, dass einer dieser Knoten mindestens Grad kleiner oder gleich sechs haben muss (sonst würde H die Reduktion R wegen der beschränkten Grad Bedingung in der Definition nicht enthalten) oder das Zentrum einer Reduktion aus CM sein. Also wurde ein Knoten x von S_R mit einem Grad kleiner oder gleich sechs oder das Zentrum einer CM Reduktion auf den Stapel gelegt. Falls $R \in EZ \cup CS_2$, dann wird diese Reduktion gefunden. Falls $R \in CM$ mit Zentrum x ist, wird diese Reduktion beim Aufruf von CENTERCHECK ebenfalls gefunden. Ansonsten ist $R \in CM$ und x ist in einer Blattstruktur enthalten, welche Teil von R ist. Da diese Blattstruktur gefunden wird und damit CENTERCHECK aufgerufen wird, wird auch in diesem Fall die Reduktion gefunden. $\mathcal{QED}$

(4.3) Satz
Gegeben sei ein Graph $G = (V, E)$ mit $|V| = n$. Der Algorithmus TREE_WIDTH_FOUR? hat eine Laufzeit von $\mathcal{O}(n)$.

Beweis
Sei G ein Graph mit n Knoten. Falls $|E(G)| > 4n$ terminiert der Algorithmus nach $4n + 1$ Schritten.

Zu beginn wird jeder Knoten auf den Stapel VS gelegt und $L(v)$ initialisiert. Die Laufzeit hierfür ist linear. Da $|E(G)| \leq 4n$ ist, ist die Laufzeit zur Markierung aller Kanten ebenfalls linear.

Zunächst zeigen wir, dass der Aufwand über den gesamten Algorithmus für das Hinzufügen von Kanten und entfernen von Mehrfachkanten linear ist. Da jede Reduktion die angewendet wird wegen Satz 3.36 eine 4-Sterne Reduktion ist, lässt sich die Anwendung einer Reduktion als Folge von Knoten-Eliminierungen von Knoten mit Grad höchstens 4 beschreiben. Das sind auch die einzigen Stellen, an denen Kanten hinzugefügt werden. Somit werden im Laufe des Algorithmus höchstens n Knoten eliminiert und, da jede Knoten-Eliminierung höchstens sechs Kanten hinzufügt, auch höchstens $6n$ Kanten hinzugefügt. Folglich werden höchstens $6n$ Mehrfachkanten erstellt und somit auch nur höchstens $6n$ Mehrfachkanten entfernt.

Die Schranke für die Anzahl der Iterationen der Hauptschleife ist implizit durch eine Schranke aller Knoten die auf VS gelegt werden gegeben. Dazu beachte man, wie im Beweis von Satz 4.2, dass höchstens n Reduktionen ausgeführt werden und somit höchstens $7n$ Knoten auf den Stapel gelegt werden, hin zuzüglich der n Knoten die zu beginn hinzugefügt werden.

Jetzt ist nur zu zeigen, dass der Zeitaufwand für jede Iteration durch eine Konstante beschränkt ist. Da beim Prüfen ob x Knoten einer Reduktion $R \in CS_2 \cup EZ$ ist nur beschränkt viele Knoten mit beschränktem Grad geprüft werden, ist wegen Lemma 4.1 der Zeitaufwand hierfür konstant. Aus dem gleichen Grund werden die Blattstrukturen $LS_1, \ldots, LS_{59}$ in konstanter Zeit gefunden. Zur Findung der $L_{i,m}$ wird für den ersten Knoten eine Laufzeit von $\mathcal{O}(m)$ benötigt. Dadurch muss aber für einen weiteren inneren Knoten aus $L_{i,m}$ diese Arbeit nicht mehr verrichtet werden. Somit kann die Arbeit auf die inneren Knoten von $L_{i,m}$ geteilt werden und ist konstant für jeden einzelnen. Mit dem gleichen Argument erhält man das die Kosten für die Verlängerung einer $L_{i,m}$ auf alle inneren Knoten konstant aufgeteilt werden kann.

Es bleibt zu zeigen, dass jeder Aufruf von CENTERCHECK nur konstant viel Zeit in Anspruch nimmt. Jeder rekursive Aufruf markiert eine neue Kante und ruft sich selbst für diese Kante und für jede Blattstruktur auf. Da nur die $L_{i,m}$ für das Größtmögliche m gespeichert wird, kann ein Knoten nur innerer Knoten einer beschränkten Anzahl von Blattstrukturen sein. Mit Lemma 3.33 erhält man, dass höchstens 20 Kanten geprüft werden müssen. Wenn dann keine $Buddy$ oder $Triple$ Reduktion gefunden wurde, dann wird die Kardinalität von A groß und die Rekursion hält an.

Somit folgt die Behauptung, da die Anzahl der Iterationen linear ist und jede Iteration konstante Zeit beansprucht. $\mathcal{QED}$

Im Beweis könnte man annehmen, das die Konstante beim Aufruf von CenterCheck groß ist, da ein Knoten in vielen Blattstrukturen enthalten sein kann. Sanders behauptet, dass ein Knoten praktisch in höchstens vier Blattstrukturen enthalten ist und somit die Konstante in diesem Algorithmus nicht groß ist, was ihn für die Praxis anwendbar macht. In Abschnitt 4.3 wird diese Behauptung anhand von diversen Beispielen überprüft. Aber zuerst werden Hinweise zur Implementierung gegeben.

4.2 Implementierung

In diesem Abschnitt wird eine grobe Struktur der Implementierung vorgestellt. Zunächst wird als Datenstruktur zur Darstellung von Graphen die adjacency_list aus den Boost C++ Libraries [1] verwendet. Diese hat den Vorteil, dass die Komplexität für Operationen, wie z.B. das Entfernen von Kanten garantiert ist. Weiter werden dadurch Iteratoren bereitgestellt, mit denen die inzidenten Kanten eines Knoten durchlaufen werden können. Auch können die Knoten und Kanten mit Eigenschaften versehen werden, wie etwa eine Markierung.

Jedoch gibt es einen Nachteil. Und zwar ändert sich die Nummerierung der Knoten, falls ein Knoten entfernt wird. Wenn der Knoten x auf den Stapel VS gelegt wird und man eine Reduktion findet, dann kann es passieren, dass der Knoten auf dem Stapel sich mit einem anderen Knoten des Graphen anstatt x identifiziert. Dies stellt jedoch dank der *Zero* Reduktion kein Problem dar. Es reicht dann aus, wenn man bei der Eliminierung eines Knotens nur die inzidenten Kanten entfernt und der Algorithmus mit einem Graphen ohne Kanten terminiert. Dadurch müssen die Knoten nicht bei der Anwendung einer Reduktion entfernt werden, sondern können zum Schluss entfernt werden.

Der Algorithmus aus Lemma 4.1 wird im folgenden vorgestellt. Dieser muss jedes mal aufgerufen werden, bevor man die Nachbarn eines Knoten prüft, da dadurch sichergestellt wird, dass der Knoten einen beschränkten Grad und keine Mehrfachkanten besitzt.

Algorithmus 4.3 Degree(x,k)

```
Initialisiere ein Array A mit k Elementen
for α ist eine Kante in x's Adjazenzliste do
  sei y ≠ x der Knoten, der inzident mit α ist
  if y ist in A enthalten then
    lösche die Mehrfachkante α
  else if A ist voll then
    return x hat Grad größer als k
  else
    füge y zu A hinzu
  end if
end for
return x hat Grad kleiner oder gleich k
```

Bei allen Reduktionen die angewendet werden handelt es sich um 4-Sterne Reduktionen. Also gilt für eine Reduktion R: $G(T_R) = G(S_R) * x_1 * \cdots * x_r$ für geeignete Knoten $x_1, \ldots, x_r$. Also kann eine Anwendung einer Reduktion durch eine Folge von Knoten-Eliminierungen dargestellt werden. Es reicht also aus, falls eine Reduktion gefunden wurde, die Knoten $x_1, \ldots, x_r$ an folgende Funktion zu übergeben.

Algorithmus 4.4 Eliminate(x)

```
for jedes Paar y, z aus der Nachbarschaft von x do
  füge eine Kante zwischen y und z ein
end for
lösche jede Kante inzident mit x
markiere x als entfernt
```

Wie oben bereits erwähnt, müssen nur die Kanten eines Knoten entfernt werden. Die Knoten vom Grad Null können anschließend nach der Hauptschleife entfernt werden. Dazu müssen wir uns merken, welche Knoten bereits entfernt wurden. Am Ende können wir einfach die nicht markierten Knoten an die Eliminationsfolge anfügen.

Bei der Implementierung ist dabei zu achten, dass bei der Überprüfung, ob zwei Knoten adjazent sind, immer die Adjazenzliste des Knoten mit beschränktem Grad durchlaufen werden muss. Ansonsten ist die Laufzeit des Algorithmus nicht linear.

Das Suchen einer Reduktion oder einer Blattstruktur kann man in zwei Schritte aufteilen. Zunächst sucht eine Funktion die möglichen inneren Knoten einer Reduktion oder einer Blattstruktur. Falls diese die Grad Bedingungen und Adjazenz Bedingungen erfüllen, dann werden diese an eine weitere Funktion übergeben. Diese Findet dann die Ankerknoten und prüft, ob es sich wirklich um eine Reduktion bzw. eine Blattstruktur handelt. Folgende zwei Algorithmen zeigen dieses Prinzip bei der Suche von LS_4.

Algorithmus 4.5 DetermineLS4(x)

```
if deg(x) = 3 then
  for y ist Nachbar von x do
    if deg(y) = 3 then
      for z ist Nachbar von x do
        if deg(z) = 3 und y und z sind nicht adjazent then
          DetermineInteriorLS4(x,y,z)
        end if
      end for
      for z ist Nachbar von y do
        if deg(z) = 3 und x und z sind nicht adjazent then
          DetermineInteriorLS4(y,x,z)
        end if
      end for
    end if
  end for
end if
```

Algorithmus 4.6 DetermineInteriorLS4(x,y,z)

```
Sei d ≠ y und d ≠ z ein Nachbar von x
Sei b ≠ x ein Nachbar von y
if z ist mit b adjazent then
  a := b
```

```
    Sei b ≠ x ein anderer Nachbar von y
  else
    Sei a ≠ x ein anderer Nachbar von y
    if a ist nicht zu z adjazent then
      return false
    end if
  end if
  Sei c ≠ x und c ≠ a ein Nachbar von z
  if alle a,b,c,d sind verschieden then
    Die Blattstruktur LS_4 wurde gefunden
    return true
  else
    return false
  end if
```

Zur Speicherung der Blattstrukturen reicht es aus die inneren Knoten und die Ankerknoten in Listen von Knoten zu Speichern. Denn es ist zum größten Teil nur notwendig zu überprüfen, ob ein Knoten ein innerer oder ein Ankerknoten einer Blattstruktur ist. Dafür müssen die Kanten der Blattstrukturen nicht gespeichert werden, was den Speicherbedarf reduziert.

Das Prüfen, ob eine Superstruktur eine Blattstruktur ist übernimmt eine Hilfsfunktion. Wie in Abschnitt 3.4 erwähnt, hilft uns die Tabelle 3.1 den Aufwand hierfür zu reduzieren. Sie beschreibt, dass Superstrukturen, die bestimmte Blattstrukturen enthalten, selbst wieder nur isomorph zu einer kleinen Anzahl von Blattstrukturen sein können. Dadurch werden viele Fallunterscheidungen überflüssig.

Falls eine Superstruktur in CenterCheck gefunden wurde, muss überprüft werden, ob dies eine Blattstruktur ist. In Algorithmus 4.1 wird die Reduktion aus BCM angewendet, falls dies nicht der Fall ist. Jedoch muss man sich vorher vergewissern, dass diese Superstruktur keine Reduktion in $CS_2 \cup EZ$ enthält. Diese Überprüfung geschieht nicht zwingend vorher. Deshalb muss man an dieser Stelle prüfen, ob das Zentrum der Superstruktur in einer solchen Reduktion enthalten ist. Falls dies auch nicht der Fall ist, dann kann man die Reduktion aus BCM anwenden.

Bis auf die erwähnte Ausnahme wurde der Algorithmus, so wie er von Sanders beschrieben wurde, implementiert. Die Ergebnisse der Analyse werden im nächsten Abschnitt beschrieben.

4.3 Analyse

Zur Analyse werden partielle k-Bäume verwendet, die in Abschnitt 2.2 vorgestellt wurden, da diese algorithmisch leicht zu erstellen sind. Einen zufälligen k-Baum erhält man, indem man zuerst eine k-Clique erstellt und dann für die restlichen $n - k$ Knoten zunächst einen zufälligen Knoten auswählt, der in einer k-Clique enthalten ist und dann Kanten zwischen den Knoten der k-Clique und dem neuen Knoten hinzufügt. Danach werden alle Kanten des Graphen durchlaufen und diese mit einer Wahrscheinlichkeit p entfernt, um einen partiellen k-Baum zu erhalten.

Der Algorithmus wurde an unterschiedlichen 4-Bäumen getestet. Unterschiedlich insofern, dass sie sich stark in der Anzahl der Kanten und in der Anzahl der Knoten unterscheiden. Es wurde die Zeit gemessen, die der Algorithmus gebraucht hat, als auch die gesamten Reduktionen gezählt, die angewandt wurden. Folgende Tabellen und Diagramme fassen die Ergebnisse zusammen. In Tabelle 4.1 wurden zu verschiedenen Wahrscheinlichkeiten p je 10000 Graphen mit je 1000 Knoten verwendet. In Tabelle 4.2 wurde die Anzahl der Knoten um den Faktor 10 erhöht, aber dafür nur jeweils 1000 Graphen betrachtet.

p	**2/3**	**1/2**	**1/3**	**1/4**	**1/6**	**0**
Zero	1449989	406043	80952	30634	14124	10000
One	2993031	1644768	585364	270991	92294	10000
Series	3025430	2928603	1931273	1283895	660954	10000
Triangle	1828458	2722820	2764959	2421235	1826061	4460
YO	490	746	327	159	33	0
H7	17454	18827	9591	4770	1473	0
TO	46	36	5	0	0	0
YI	2193	1502	428	144	29	0
L1	3	2	0	0	0	0
L2	8	6	1	0	0	0
L3	18	16	4	1	0	0
L4	83	114	54	13	2	0
BCM	132015	450549	889021	1124081	1361685	1906605
Buddy	95241	189228	206055	170670	112214	0
Triple	40626	176458	431895	599996	772231	889687
∅ Zeit (ms)	205	304	392	429	456	502

Tabelle 4.1: Ergebnisse für 10000 4-Bäume mit 1000 Knoten

p	**2/3**	**1/2**	**1/3**	**1/4**	**1/6**	**0**
Zero	1449953	401238	73137	21985	5310	1000
One	3018187	1658682	584233	266147	84762	1000
Series	3016243	2940219	1944771	1291375	661851	1000
Triangle	1796951	2672875	2726930	2390180	1807885	415
YO	561	716	367	136	28	0
H7	17298	18486	9529	4758	1472	0
TO	46	41	5	3	0	0
YI	2084	1510	501	126	25	0
L1	5	0	0	0	0	0
L2	7	5	2	2	0	0
L3	26	15	3	0	1	0
L4	68	95	47	25	11	0
BCM	136603	451420	876139	1104347	1334225	1867047
Buddy	96418	195216	212407	177890	116079	0
Triple	40764	180663	448396	623220	803976	933413
∅ Zeit (ms)	2145	3206	4122	4707	5058	5584

Tabelle 4.2: Ergebnisse für 1000 4-Bäume mit 10000 Knoten

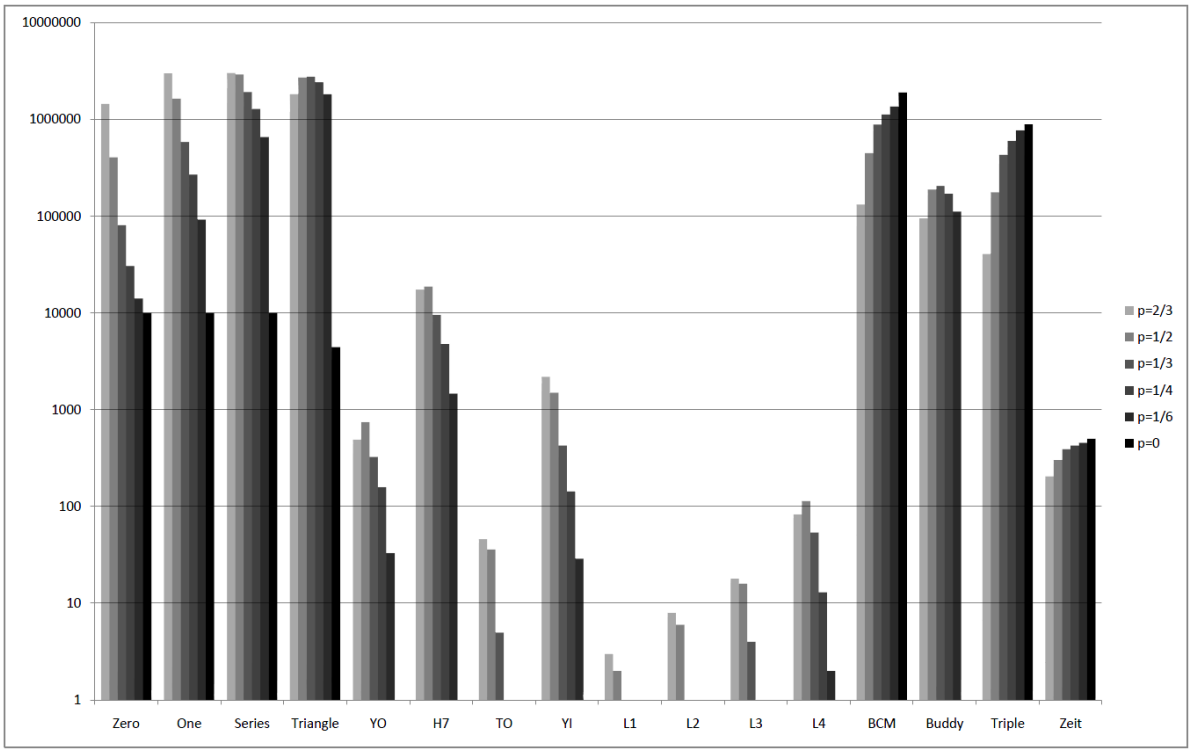

Abbildung 4.1: Diagramm zu Tabelle 4.1

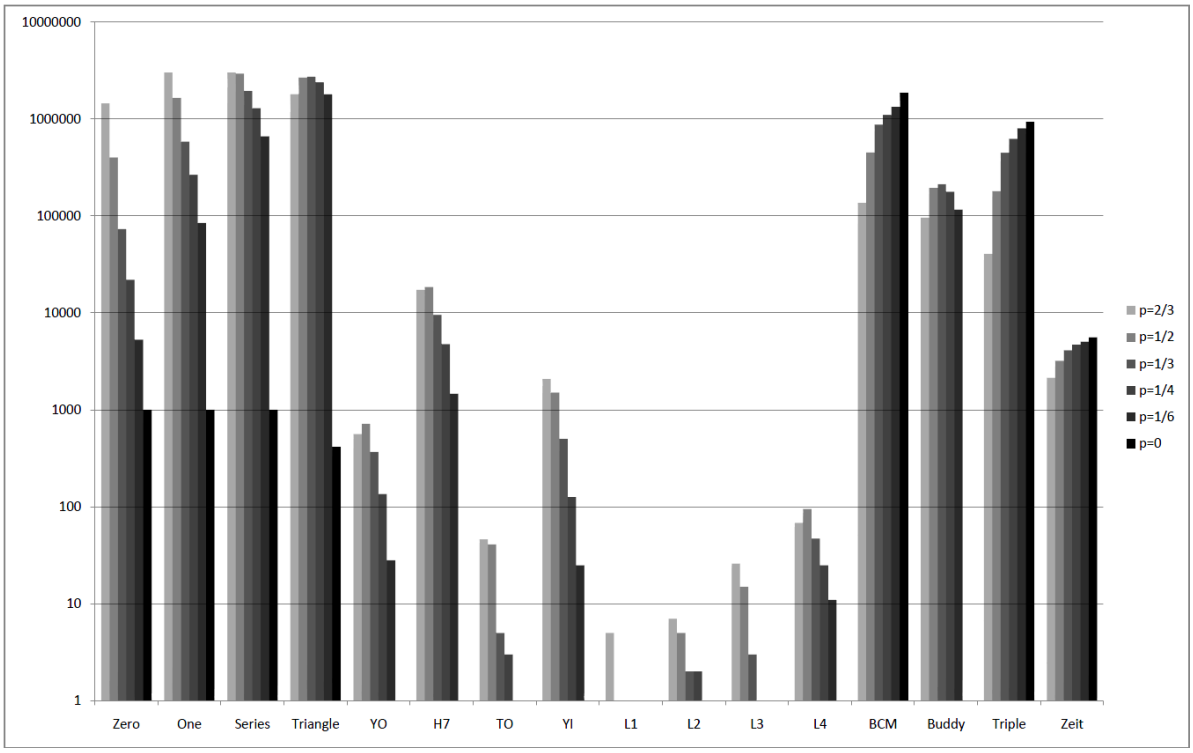

Abbildung 4.2: Diagramm zu Tabelle 4.2

Man sieht deutlich, dass die Anzahl der Reduktionen aus $CS_2 \cup EZ$ sinkt, je dichter der Graph wird, wohingegen die Anzahl der Reduktionen aus CM steigt. Diese Beobachtung kann man damit erklären, dass wegen der höheren Anzahl an Kanten weniger Knoten mit einem Grad kleiner oder gleich drei existieren. Dadurch wird es gleichzeitig Wahrscheinlicher, dass eine Superstruktur den vollständigen Graphen als Minor enthält. Auffällig ist jedoch, dass die Reduktionen aus EZ, außer der *Triangle* Reduktion, allgemein selten gefunden werden. Das liegt auch daran, dass die Strukturen sehr speziell sind. Somit werden diese nur wegen der Vollständigkeit betrachtet.

Getestet wurde der Algorithmus auf einem Notebook mit zwei 2.5 GHz Prozessoren, 4Gb RAM und einem Windows 7 64-Bit Betriebssystem. Der Zeitaufwand ist selbst für große Graphen gering, wie von Sanders behauptet, und der Algorithmus ist somit in der Praxis anwendbar.

Die Ergebnisse für partielle 4-Bäume sind überaus zufriedenstellend. Deswegen wird der Algorithmus an einem speziellen Beispiel getestet, nämlich am Petersen Graphen. Der Graph wird in Abbildung 4.3 dargestellt. Bis jetzt war nur bekannt, dass der Graph eine Baumweite von vier besitzt. Wir werden sehen, wie dieser durch Reduktionen beschrieben wird. Bei der Anwendung des Algorithmus stellt sich raus, dass dieser im ersten Schritt auf den vollständigen Graphen auf vier Knoten durch eine BCM Reduktion reduziert wird. Abbildung 4.3 zeigt auch, wie der Petersen Graph aus Blattstrukturen zusammengesetzt ist. Die letzten Reduktionen sind dann die

Triangle, *Series*, *One*, *Zero*, die den Graphen auf den letzten vier Knoten in den leeren Graphen überführen.

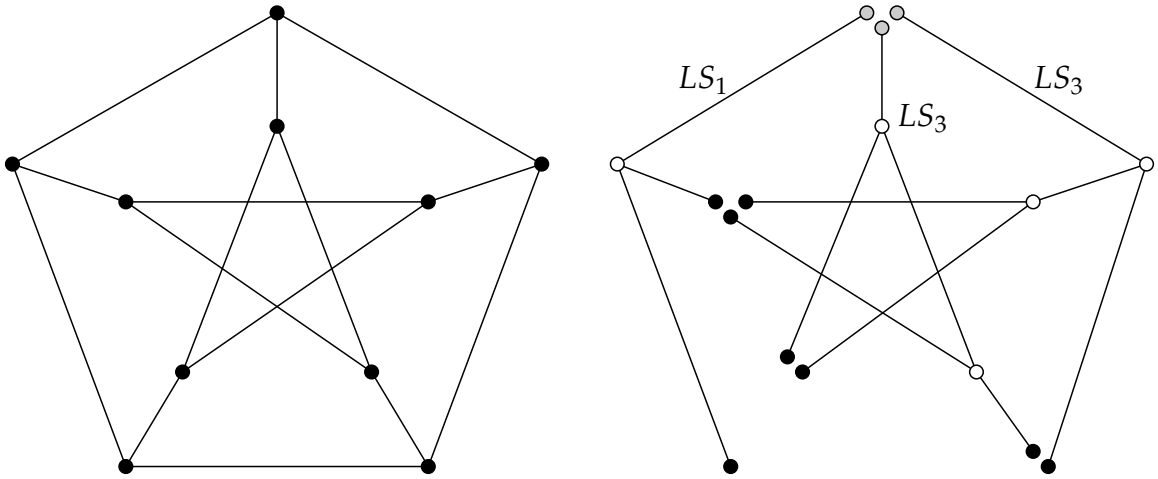

Abbildung 4.3: Petersen Graph

Zum Schluss können wir noch testen, wieweit dieser Algorithmus partielle k-Bäume für $k > 4$ reduzieren kann. Es ist klar, dass ein k-Baum für $k > 4$ keine einzige Reduktion in CS_4 besitzt, da jeder Knoten mindestens Grad fünf hat. Aus diesem Grund werden diese auch nicht betrachtet. Wir testen jeweils 10000 partielle k-Bäume mit 1000 Knoten, für $k = 5$, $k = 7$ und $k = 10$. Dabei werden die gleichen Wahrscheinlichkeiten p zur Entfernung einer Kante wie bei den partiellen 4-Bäumen verwendet. Es wurde wieder die Zeit gemessen und die Anzahl der Reduktionen gezählt. Zusätzlich sehen wir uns an, wie viele Knoten und Kanten letztendlich durch Reduktionen entfernt wurden. Die nächsten drei Diagramme veranschaulichen die Ergebnisse.

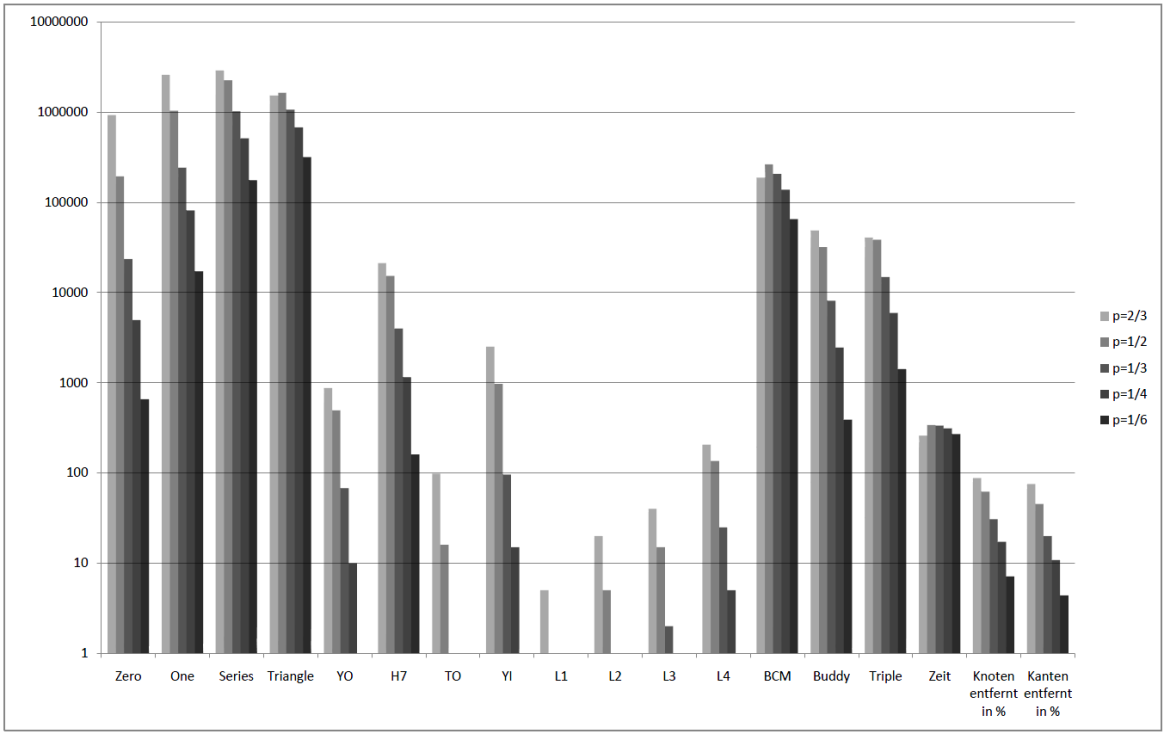

Abbildung 4.4: Diagramm zu partiellen 5-Bäumen

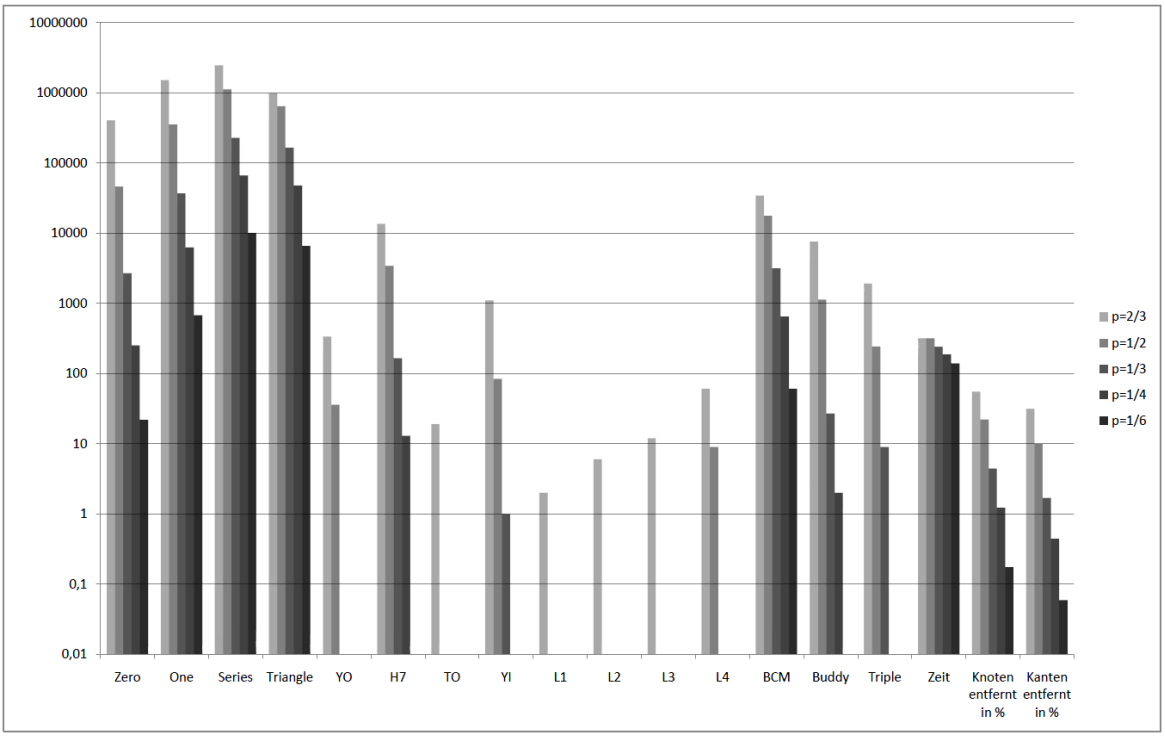

Abbildung 4.5: Diagramm zu partiellen 7-Bäumen

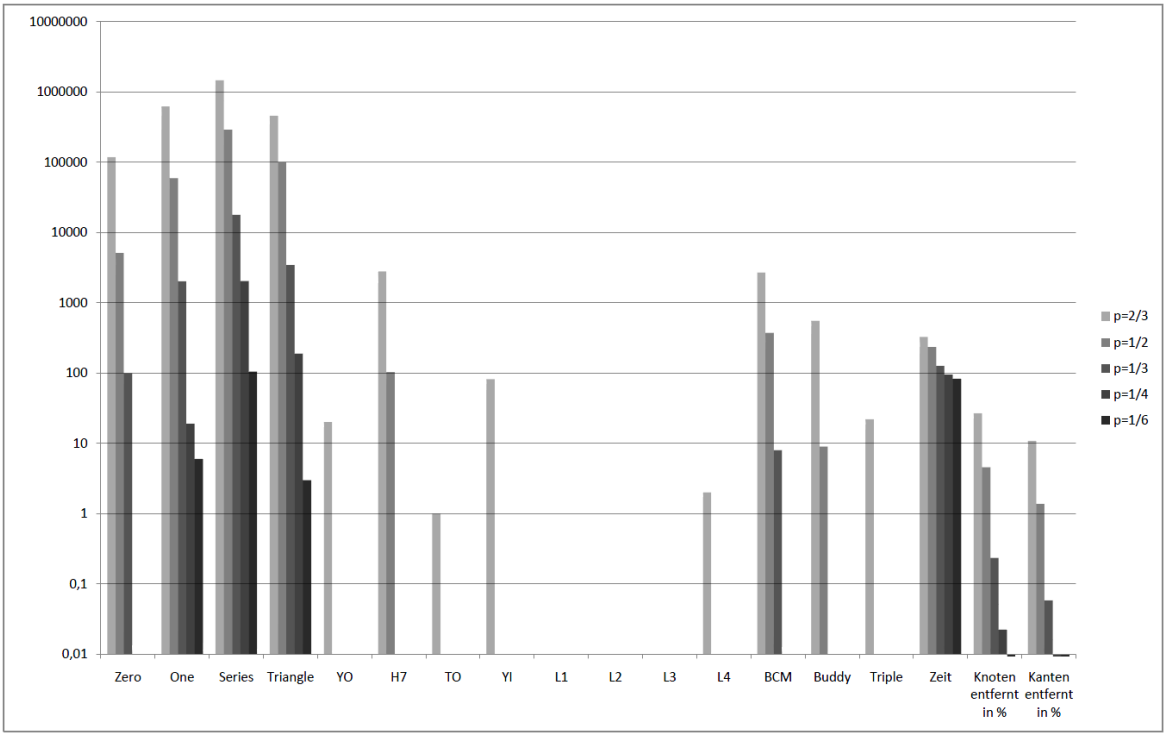

Abbildung 4.6: Diagramm zu partiellen 10-Bäumen

Anhand dieser Daten kann man erkennen, dass der Algorithmus auch viele Graphen mit Baumweite k, wobei k nahe bei vier liegt, auf kleinere Graphen reduzieren kann. Die Anzahl der entfernten Knoten und Kanten hängt von der Anzahl der Kanten des Graphen und von k ab, also von der Dichte des Graphen ab. Dabei hat eine kleinere Dichte eine erhöhte Anzahl an Reduktionen zufolge. Leider kann man feststellen, dass die Reduktionen aus $CM \cup EZ \setminus \{Triangle\}$ den Reduktionen aus $CS_2 \cup \{Triangle\}$ deutlich unterlegen sind. Aus diesem Grund bietet der Algorithmus nur eine kleine Verbesserung für Graphen mit größerer Baumweite, gegenüber bekannten Algorithmen, die mithilfe von Reduktionen prüfen, ob ein Graph Baumweite kleiner oder gleich drei besitzt.

4.4 Ausblick

Diese Arbeit zeigt, dass mithilfe von geeigneten Algorithmen viele NP-vollständige Probleme für Graphen mit Baumweite kleiner oder gleich vier in linearer Zeit gelöst werden können. Die Ergebnisse aus Abschnitt 4.3 zeigen, dass der Algorithmus keine große Konstante besitzt und der Algorithmus in der Praxis auch auf große Graphen anwendbar ist.

Man könnte versuchen diese Arbeit auf Graphen mit Baumweite kleiner oder gleich fünf zu

erweitern. Dabei Werden die Superstrukturen auf maximal fünf Ankerknoten erweitert. Das Verfahren aus Abschnitt 3.4 zur Konstruktion der Blattstrukturen könnte man wieder anwenden. Dazu müsste man zunächst Superstrukturen bilden und versuchen in diesen sichere Reduktion zu finden, bis man eine Menge von Blattstrukturen auf maximal fünf Knoten gefunden hat, die wieder die Bedingung erfüllen, dass die Superstrukturen, die man daraus bilden kann entweder wieder Blattstrukturen sind, oder eine Reduktion enthalten. Jedoch kann man davon ausgehen, dass dies einen noch größeren Aufwand erfordert, wie der Schritt von $k = 3$ nach $k = 4$ schon gezeigt hat.

Literaturverzeichnis

[1] DAWES, Beman (Hrsg.) ; ABRAHAMS, David (Hrsg.): *Boost C++ Libraries*. – URL `http://www.boost.org/`

[2] ARNBORG, Stefan: Reduced state enumeration–another algorithm for reliability evaluation. In: *IEEE Trans. Reliability* 27 (1978), S. 101 – 105

[3] ARNBORG, Stefan: Efficient algorithms for combinatorial problems on graphs with bounded decomposability — A survey. In: *BIT Numerical Mathematics* 25 (1985), Nr. 1, S. 1 – 23

[4] ARNBORG, Stefan ; PROSKUROWSKI, Andrzej: Characterization and Recognition of Partial 3-Trees. In: *SIAM Journal on Algebraic and Discrete Methods* 7 (1986), Nr. 2, S. 305–314

[5] ARNBORG, Stefan ; PROSKUROWSKI, Andrzej: Linear time algorithms for NP-hard problems restricted to partial k-trees. In: *Discrete Applied Mathematics* 23 (1989), Nr. 1, S. 11 – 24. – ISSN 0166-218X

[6] ARNBORG, Stefan ; PROSKUROWSKI, Andrzej ; CORNEIL, Derek G.: Forbidden minors characterization of partial 3-trees. In: *Discrete Mathematics* 80 (1990), Nr. 1, S. 1 – 19. – ISSN 0012-365X

[7] BODLAENDER, Hans L.: A partial k-arboretum of graphs with bounded treewidth. In: *Theoretical Computer Science* 209 (1998), Nr. 1-2, S. 1 – 45. – ISSN 0304-3975

[8] BODLAENDER, Hans L. ; KOSTER, Arie M. C. A.: Combinatorial Optimization on Graphs of Bounded Treewidth. In: *Comput. J.* 51 (2008), Nr. 3, S. 255–269

[9] BODLAENDER, Hans L. ; KOSTER, Arie M. C. A. ; EIJKHOF, Frank van den: Preprocessing Rules for Triangulation of Probabilistic Networks. In: *Computational Intelligence* 21 (2005), Nr. 3, S. 286–305

[10] DIESTEL, Reinhard: *Graphentheorie*. 3. Auflage. Springer Verlag, Heidelberg, 2006. – ISBN 3-540-21391-0

[11] EIJKHOF, Frank van den ; BODLAENDER, Hans L. ; KOSTER, M. C. A.: Safe Reduction Rules for Weighted Treewidth. In: *Algorithmica* 47 (2007), Nr. 2, S. 139–158. – ISSN 0178-4617

[12] HEGGERNES, Pinar: Treewidth, partial k-trees, and chordal graphs / Department of Informatics, University of Bergen, Norway. 2005. – Partial curriculum in INF334 - Advanced algorithmical techniques

[13] KAJITANI, Yoji ; ISHIZUKA, Akio ; UENO, Shuichi: Characterization of partial 3-trees in terms of three structures. In: *Graphs and Combinatorics* 2 (1986), Nr. 1, S. 233 – 246

[14] LAGERGREN, Jens: The nonexistence of reduction rules giving an embedding into a k-tree. In: *Discrete Applied Mathematics* 54 (1994), Nr. 2-3, S. 219 – 223. – ISSN 0166-218X

[15] ROSE, Donald J.: On simple characterizations of k-trees. In: *Discrete Mathematics* 7 (1974), Nr. 3-4, S. 317 – 322. – ISSN 0012-365X

[16] SANDERS, Daniel P.: *Linear Algorithms for Graphs of Tree-width at Most Four*. Atlanta, Georgia, Program of Algorithms, Combinatorics, and Optimization, Georgia Institute of Technology, Dissertation, 1993

[17] SANDERS, Daniel P.: On Linear Recognition of Tree-Width at Most Four. In: *SIAM J. Discret. Math.* 9 (1996), Nr. 1, S. 101–117. – ISSN 0895-4801

[18] URBAN, Karsten: *Eine Einführung in C++*. Aachen : Shaker Verlag, 1998. – ISBN 10: 3826537491

Stichwortverzeichnis

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Aachen, 22. September 2010 .....

Alexander Hein