Effiziente parallele Algorithmen für Bäume und modulare Erweiterungen

Dem Fachbereich 11 (Mathematik und Informatik) der

Gerhard–Mercator–Universität — Gesamthochschule Duisburg

vorgelegte Diplomarbeit zur Erlangung des akademischen Grades

Dipl.-Math.

von

Thomas Szymczak

aus

Dinslaken (NRW)

Gutachter: Prof. Dr. Andreas Brandstädt
Zweitgutachter: (wird vom Prüfungsausschuß bestimmt)

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einführung

1.1 Einleitung und Gliederung

Viele Probleme, nicht nur aus der Informatik, lassen sich durch Graphen beschreiben. Klassische Beispiele hierfür sind u.a. das Stundenplanproblem (Problem der (zeitlichen) Einteilung der Lehrer einer Schule auf die Klassen), das Vier–Farben Problem (d.h. man möchte entscheiden, ob sich für jede Landkarte die Länder so mit vier Farben färben lassen, daß Länder mit einer gemeinsamen Grenze eine verschiedene Farbe erhalten) und Transportprobleme.

Doch für die meisten Probleme auf Graphen ist eine effiziente (polynomiale) Lösbarkeit nicht zu erwarten, da sie $\mathbb{NP}$–vollständig sind. Daher hat man im Prinzip nur zwei Möglichkeiten:

1. Entwicklung von Approximationsheuristiken, d.h. man möchte polynomiell eine (möglichst gute) Näherungslösung für das gegebene Problem bestimmen.

2. Einschränkung der Eingabe, d.h. man gibt einen Algorithmus an, der das Problem effizient für Graphen einer speziellen Graphenklasse löst.

Die zweite Möglichkeit führte u.a. zur Einführung einer Fülle von Graphenklassen. Einen Überblick über etwa 120 Graphenklassen und deren Inklusionen findet man in dem Survey [Bra93].

Viele effiziente Algorithmen beruhen darauf, daß man einen unterliegenden Baum vorliegen hat (u.a. Heapsort, Suchbäume). Daher ist man zum einen bestrebt, effiziente Algorithmen für Bäume zu entwickeln, und zum anderen möchte man Graphenklassen mit einer gewissen Baumstruktur näher untersuchen.

Diese beiden Wege beschreiten wir auch in dieser Arbeit. Zunächst geben wir im ersten Kapitel die grundlegenden Bezeichnungen aus der Graphen– und Komplexitätstheorie an, die wir in den folgenden Kapiteln benötigen werden.

Anschließend, im zweiten Kapitel, beschreiben wir, wie man parallel effizient algebraische Ausdrücke auf Bäumen auswerten kann. Grundlage hierfür ist der optimale Baum–Kontraktionsalgorithmus von ABRAHAMSON ET ALTERA aus [ADKP89], der in Abschnitt 2.2 erklärt wird.

Im dritten Kapitel zeigen wir dann, daß sich einige Probleme auf Bäumen auf die Auswertung von algebraischen Ausdrücken zurückführen lassen. Unter Ausnutzung der Ergebnisse des

zweiten Kapitels erhalten wir somit effiziente parallele Algorithmen für diese Probleme. Unser Schwerpunkt liegt dabei auf den Dominationsproblemen.

Es sei $G = (V, E)$ ein Graph. Eine Menge $D \subseteq V$ heißt eine dominierende Menge, falls jeder Knoten $v \in V \setminus D$ zu einem Knoten aus D adjazent ist. Das **DS**–Problem besteht nun darin, eine (bzgl. der Kardinalität) minimale dominierende Menge zu berechnen. Es existieren viele Abwandlungen bzw. Verallgemeinerungen dieses **DS**–Problems, eine Übersicht einiger dieser Dominationsprobleme findet der Leser in Abschnitt 3.4.

Im letzten Kapitel beschäftigen wir uns mit modularen Erweiterungen von Graphen (bzw. Graphenklassen). Seien G und H zwei Graphen sowie v ein Knoten in G. Dann sei $\mathrm{MExt}(G, v, H)$ derjenige Graph, den man erhält, wenn man den Graphen H „nachbarschaftstreu“ in den Knoten v (in G) einsetzt. Graphen, die durch endlichmalige Anwendung dieser „Operation“ aus G hervorgehen, bezeichnen wir als modulare Erweiterungen von G. (Eine mathematisch exakte Definition geben wir in Abschnitt 4.1 an. Dort wird auch die Namensgebung motiviert.) In den ersten drei Abschnitten dieses vierten Kapitels leiten wir zunächst einige, für die folgenden Abschnitte benötigte, theoretische Ergebnisse für modulare Erweiterungen und verwandte Begriffe her. In Abschnitt 4.4 zeigen wir dann, daß sich modulare Erweiterungen einer Graphenklasse **G**, die bezüglich Primgraphbildung abgeschlossen ist, genauso effizient sequentiell und parallel erkennen lassen, wie Graphen aus **G**. Hierbei erhalten wir auch eine Baumstruktur–Charakterisierung für modulare Erweiterungen von Bäumen.

Es gibt mehrere Möglichkeiten, eine Baumstruktur auf einer Graphenklasse zu definieren. So lassen sich z.B. recht viele Graphenklassen **G** über Hyperbäume charakterisieren, d.h. man kann jedem Graphen G einen Hypergraphen $\mathcal{H}(G)$ so zuordnen, daß gilt: $\mathcal{H}(G)$ ist genau dann ein Hyperbaum (bzw. dualer Hyperbaum), wenn $G \in \mathbf{G}$ gilt.

Die bekanntesten Beispiele hierfür sind die chordalen, dual chordalen und distanz–erblichen Graphen:

Satz 1.1.1
Es gelten folgende Aussagen:

(1) ([Wal72], [Gav74], [Bun74])
Ein Graph G ist genau dann chordal, wenn der Hypergraph $\mathcal{C}(G)$ der maximalen Cliquen in G ein dualer Hyperbaum ist.

(2) ([BDCV93])
Folgende Aussagen sind für einen Graphen G äquivalent:

- *(a) G ist dual chordal.*
- *(b) $\mathcal{C}(G)$ ist ein Hyperbaum.*[1]
- *(c) Der Hypergraph $\mathcal{N}(G)$ der (abgeschlossenen) Nachbarschaften in G ist ein Hyperbaum.*
- *(d) $\mathcal{N}(G)$ ist ein dualer Hyperbaum.*

(3) ([Nic94b], [Nic94a])
Ein Graph G ist genau dann distanz–erblich, wenn der Hypergraph der maximalen zusammenhängenden Cographen in G ein dualer Hyperbaum ist. ■

[1] Diese Eigenschaft zusammen mit (1) führte zur Namensgebung der „dual“ chordalen Graphen.

Bei modularen Erweiterungen von Bäumen (für diese Klasse schreiben wir im folgenden kurz $\mathrm{MExt}^*(\textbf{Bäume})$) erhält man eine noch „direktere“ Baumstruktur, eine sogenannte Baumdarstellung. Wir zeigen in Abschnitt 4.4, daß sich jeder Graph $G \in \mathrm{MExt}^*(\textbf{Bäume})$ wie folgt aus einem Baum erzeugen läßt: Es gibt einen Baum $T = (V, E)$ und eine Familie $(H_v)_{v \in V}$ von Graphen so, daß derjenige Graph, den man erhält, wenn man (nacheinander) für jedes $v \in V$ den Graphen H_v „nachbarschaftstreu“ in v einsetzt, gleich G ist. (Man beachte, daß eine solche Baumdarstellung nicht sofort aus der Definition der modularen Erweiterungen zu erwarten ist.) Diese Baumstruktur nutzen wir im Abschnitt 4.5 aus, um einen sequentiellen Algorithmus zu entwickeln, der das **RDS**–Problem (eine Verallgemeinerung des **DS**–Problems) auf modularen Erweiterungen von Bäumen in Linearzeit löst. Der Algorithmus (bzw. der Beweis für die Korrektheit des Algorithmus) ist recht kompliziert und erfordert mehrfache Fallunterscheidungen, was zum Teil daran liegt, daß die Graphen, „die in dem unterliegenden Baum der Baumdarstellung eingesetzt sind“, beliebige Graphen sein können.

Die effiziente Lösbarkeit des **RDS**–Problem auf $\mathrm{MExt}^*(\textbf{Bäume})$ liegt auch an der „Natur“ der Dominationsprobleme (insbesondere in Hinblick auf Moduln). Denn, wie wir in Abschnitt 5.6 zeigen werden, sind viele der klassischen Graphenprobleme auf $\mathrm{MExt}^*(\textbf{Bäume})$ $\mathbb{NP}$–vollständig (u.a. die Berechnung der Graphenparameter, das HAMILTONkreis– und HAMILTONpfad–Problem). Dies liegt u.a. daran, daß die Graphen aus $\mathrm{MExt}^*(\textbf{Bäume})$ im allgemeinen nicht perfekt sind (siehe Abschnitt 5.6).

1.2 Grundlagen und Definitionen

1.2.1 Mathematische Bezeichnungen

Wir geben hier nur diejenigen Bezeichnungen an, die häufig von Autor zu Autor variieren. Weitere Grundbegriffe setzen wir voraus.

Die natürlichen Zahlen (bzw. ganzen Zahlen) bezeichen wir mit $\mathbb{N}$ (bzw. $\mathbb{Z}$). In unserem Sinne ist 0 eine natürliche Zahl.

Ist M eine Menge und sind A, B zwei Teilmengen von M, so schreiben wir

- $|M|$ für die *Kardinalität* von M,
- $A \subseteq B$, falls A eine Teilmenge von B ist,
- $A \subset B$, falls $A \subseteq B$ und $A \neq B$ erfüllt ist.

Es seien M eine Menge und $k \in \mathbb{N}$. Dann bezeichnen wir mit $\mathcal{P}_k(M)$ die *Menge aller k–elementigen Teilmengen von M*. Die *Potenzmenge* $\mathcal{P}(M)$ von M ist dann gegeben durch $\mathcal{P}(M) = \bigcup_{k \in \mathbb{N}} \mathcal{P}_k(M)$.

Zur Komplexitätsbeschreibung benutzen wir die LANDAUschen Symbole: Seien $f, g : \mathbb{N} \to \mathbb{N}$ zwei Funktionen. Dann schreiben wir $f \in O(g)$, falls es natürliche Zahlen $C, n_0 \in \mathbb{N}$ so gibt, daß $f(n) \leq C \cdot g(n)$ für alle $n \geq n_0$ erfüllt ist.

1.2.2 Graphentheoretische Grundlagen

Es seien V eine endliche Menge und $E \subseteq \mathcal{P}_2(M)$. Dann nennen wir das Paar $G = (V, E)$ einen *Graphen*. Die Elemente aus V heißen *Knoten* und jedes $e \in E$ nennt man eine *Kante*. Graphen lassen sich durch Angabe von Inzidenzen in allgemeinerer Form definieren (vgl. [Bra94]). In diesem Sinne sind unsere Graphen ungerichtet, einfach (d.h. ohne Doppelkanten und Schlingen) und endlich.

Für eine Kante $\{x, y\} \in E$ schreiben wir kurz xy. x und y heißen in diesem Fall *benachbart* (oder *adjazent*).

Ist $U \subseteq V$, so bezeichnen wir mit $G(U)$ den von U in G induzierten Teilgraphen, d.h. $G(U) := (U, \mathcal{P}_2(U) \cap E)$. Für $G(V \setminus U)$ schreiben wir kurz $G - U$ und $G - u$, falls $U = \{u\}$ gilt.

Mit $\overline{G}$ bezeichnen wir den zu G *komplementären Graphen*, d.h. $V(\overline{G}) := V$, $E(\overline{G}) := \mathcal{P}_2(V) \setminus E$.

G heißt *vollständig*, falls $E = \mathcal{P}_2(V)$. G heißt *unabhängig*, falls $\overline{G}$ vollständig ist. Einen vollständigen Graphen mit n Knoten bezeichnen wir kurz mit K_n. Ist $U \subseteq V$, so nennen wir U eine vollständige Menge[2] (bzw. unabhängige Menge) in G, falls $G(U)$ vollständig (bzw. unabhängig) ist.

Sind $G_i = (V_i, E_i)$, $i = 1, 2$, zwei Graphen mit $V_1 \cap V_2 = \emptyset$, so bezeichnet $G_1 + G_2 = (V', E')$ die *disjunkte Vereinigung* von G, d.h. $V' = V_1 \cup V_2$ und $E' = E_1 \cup E_2$.[3] Ist G ein Graph und $k \in \mathbb{N}$, $k \geq 2$, so sei $kG := \underbrace{G + \ldots + G}_{k \text{ mal}}$.

Für $v \in V$ sei $N(v) := \{w \in V : vw \in E\}$ die *(offene) Nachbarschaft* und $N[v] := N(v) \cup \{v\}$ die *abgeschlossene Nachbarschaft* von v. Die Anzahl der Knoten aus $N(v)$ ist der *Grad* $\deg(v)$ von v. Dann gilt $2|E| = \sum_{v \in V} \deg(v)$ (*Handschlagslemma*). Ist $U \subseteq V$, so sei

$$N[U] := \bigcup_{u \in U} N[u] \qquad \text{(bzw. } N(U) := N[U] \setminus U\text{)}$$

die *(offene) Nachbarschaft* (bzw. *abgeschlossene Nachbarschaft*) von U.

Die *k–te Nachbarschaft* $N^k(v)$ eines Knotens v sei die Menge aller Knoten aus V, die von v den Abstand k besitzen, d.h.

$$N^k(v) := \{w \in V : d(v, w) = k\}.$$

Die *k–te Disk* $D(v, k)$ um v sei die Menge aller Knoten aus V, die von v einen Abstand kleiner oder gleich k haben, also

$$D(v, k) := \{w \in V : d(v, w) \leq k\} = \bigcup_{i=0}^{k} N^k(v).$$

Ein *Pfad* von v_0 (*Anfangsknoten*) nach v_k (*Endknoten*) in G ist eine Familie $(v_0, v_1, \ldots, v_k)$ von paarweise verschiedenen Knoten (bis auf evtl. $v_0 = v_k$), so daß für alle $i \in \{0, \ldots, k-1\}$

[2] Eine vollständige Menge in einen Graphen nennt man auch eine *Clique*.

[3] Sind V_1, V_2 nicht disjunkt, so definiert man $G_1 + G_2$, indem man V_1 mit $V_1 \times \{1\}$ und V_2 mit $V_2 \times \{2\}$ identifiziert.

gilt $v_iv_{i+1} \in E$ (Kurzschreibweise hierfür: $v_0 - v_1 - \ldots - v_k$). Die Zahl k der Kanten des Pfades nennen wir die *Länge des Pfades*. Ein *Kreis* in G ist ein Pfad mit gleichem Anfangs– und Endknoten. Ein Pfad (bzw. Kreis) $(v_0, \ldots, v_k)$ heißt *sehnenlos*, wenn $v_iv_j \notin E$ für alle $i, j \in \{0, \ldots, k\}$ mit $|i - j| \geq 2$ (bzw. $v_iv_j \notin E$ für alle $i, j \in \{0, \ldots, k-1\}$ mit $|i - j| \geq 2$ $(\mathrm{mod}\ k-1)$). Sehnenlose Pfade (bzw. Kreise) der Länge $n-1$ (bzw. n), d.h. mit n Knoten, bezeichnen wir mit P_n (bzw. C_n).

Einen Pfad (bzw. Kreis) der Länge $|V|-1$ (bzw. $|V|$) in G nennen wir einen HAMILTONpfad (bzw. HAMILTONkreis).

Sind $v, w \in V$, so sei mit $d_G(v, w)$ der *Abstand von v zu w* gemeint, das ist die minimale Länge eines Pfades von v nach w, falls ein solcher existiert. Existiert kein solcher Pfad, so setzen wir $d_G(v, w) := \infty$. Oft lassen wir auch den Index G weg, wenn keine Verwechslungen zu befürchten sind. Ist $v \in V$ und $W \subseteq V$ eine nichtleere Teilmenge von V, so sei

$$d(v, W) := \min_{w \in W} d(v, w).$$

G heißt *zusammenhängend*, wenn zwischen je zwei Knoten aus V ein Pfad existiert, der beide Knoten verbindet. $W \subseteq V$ heißt eine *Zusammenhangskomponente* von G, wenn $G(W)$ zusammenhängend ist und für jedes $v \in V \backslash W$ der Graph $G(W \cup \{v\})$ nicht zusammenhängend ist. Die Zusammenhangskomponenten von G bezeichnen wir mit $\mathrm{ZHK}(G)$ und die Anzahl der Zusammenhangskomponenten von G mit $c(G)$. G heißt *kreisfrei*, wenn G keinen C_n, $n \geq 3$, als induzierten Teilgraphen enthält. Kreisfreie Graphen bezeichnet man auch als *Wälder*.

Ein Graph $T = (V, E)$ heißt *Baum* (oder auch *freier Baum*), wenn er zusammenhängend und kreisfrei ist. Ist zusätzlich ein bestimmter Knoten $w \in V$ ausgezeichnet, so nennen wir T einen *Wurzelbaum* mit *Wurzel* w. Wenn wir zum Ausdruck bringen möchten, daß es sich bei einem Baum T um einen Wurzelbaum mit Wurzel w handelt, so schreiben wir T_w für T. Die Knoten aus T, die Grad 1 besitzen, nennt man die *Blätter* in T.

In Bäumen gibt es bekanntlich für je zwei Knoten $u, v \in V$ genau einen Pfad, der beide Knoten verbindet (genauer sind Bäume sogar durch diese Eigenschaft charakterisiert). Diesen Pfad bezeichnen wir im folgenden mit $P(u, v)$.

Sei $T_w = (V, E)$ ein Wurzelbaum und $v \in V$. Jeden Knoten aus $P(v, w)$ nennt man einen *Vorgänger* von v und jeden Knoten aus $\{u \in V \setminus \{v\} : v \in P(u, w)\}$ einen *Nachfolger* von v. Vorgänger bzw. Nachfolger von v, die zu v benachbart sind, nennt man *unmittelbare Vorgänger* bzw. *unmittelbare Nachfolger*. Wegen der Kreisfreiheit hat jeder Knoten (außer w) genau einen unmittelbaren Vorgänger $\mathrm{parent}(v)$. Häufig bezeichnet man den unmittelbaren Vorgänger auch als *Vater* und die unmittelbaren Nachfolger als *Söhne* oder *Kinder*. Die Anzahl der unmittelbaren Nachfolger eines Knotens v nennen wir den *Ausgangsgrad* $\mathrm{outdeg}(v)$ und den Abstand von v zur Wurzel die *Tiefe* $\mathrm{depth}(v)$ von v. Die Tiefe $\mathrm{depth}(T)$ von T sei durch $\mathrm{depth}(T) := \max_{v \in V} \mathrm{depth}(v)$ definiert.

Ein Graph $G = (V, E)$ sei für uns in Adjazenzlistendarstellung gegeben, d.h. G wird beschrieben durch

- eine lineare Liste, welche die Knoten aus G in einer beliebigen Reihenfolge enthält,
- für jeden Knoten $v \in V$ sei eine (doppelt verkettete) lineare Liste (*Adjazenzliste* zu v) vorhanden, worin die Knoten aus $N(v)$ abgespeichert sind.

Für Wurzelbäume wählt man eine spezielle Darstellung. Für jeden Knoten v sei anstatt der Adjazenzliste ein Zeiger auf den unmittelbaren Vorgänger und eine (doppelt verkettete) lineare Liste der unmittelbaren Nachfolger gegeben.

Mit T_v bezeichen wir den von v und seinen Nachfolgern induzierten Wurzelbaum mit Wurzel v.

Nun noch einige Bezeichnungen aus der Theorie der Hypergraphen. Ein Tupel $\mathcal{H} := (V, \mathcal{E})$ heißt *Hypergraph*, wenn $\mathcal{E} \subseteq \mathcal{P}(V)$. Häufig identifizieren wir $\mathcal{H}$ mit $\mathcal{E}$. Mit $\mathcal{E}(v)$ meinen wir die Menge aller Hyperkanten aus $\mathcal{E}$, die den Knoten v enthalten. Ist $\mathcal{H} = (V, \mathcal{E})$ ein Hypergraph, so heißt $\mathcal{H}^* := (\mathcal{E}, \{\mathcal{E}(v) : v \in V\})$ der zu $\mathcal{H}$ *duale Hypergraph*.

Ein Hypergraph $\mathcal{H} = (V, \mathcal{E})$ heißt *Hyperbaum*, wenn ein Baum T mit Knotenmenge V so existiert, daß $T(e)$ für jedes $e \in \mathcal{E}$ zusammenhängend ist. Ein Hypergraph $\mathcal{H}$ heißt *dualer Hyperbaum*, wenn $\mathcal{H}^*$ ein Hyperbaum ist, d.h. es existiert ein Baum T mit Knotenmenge $\mathcal{E}$, so daß $T(\mathcal{E}(v))$ für jedes $v \in V$ zusammenhängend ist. T nennen wir dann einen *unterliegenden Baum* von $\mathcal{H}$.

1.2.3 Grundlagen für parallele Algorithmen

Als Modell für die parallelen Algorithmen in dieser Arbeit benutzen wir die **PRAM** *(parallel random access machine)*. Sie besteht aus einer gewissen Anzahl von (synchron arbeitenden) identischen Prozessoren (für die einzelnen Prozessoren benutzt man das Standardmodell der **RAM**, siehe etwa [AHU74]) und einem gemeinsamen Speicher (*shared memory*). In jedem Ausführungsschritt kann ein Prozessor eine Lese– oder Schreiboperation auf eine Speicherzelle oder eine Grundrechenfunktion durchführen. Hierbei kann es natürlich vorkommen, daß in einem Ausführungsschritt mehrere Prozessoren auf dieselbe Speicherzelle eine Lese– oder Schreiboperation ausführen möchten (*Speicherkonflikte*). Je nachdem, wie die Speicherkonflikte behandelt werden, unterscheidet man verschiedene Typen von **PRAM**s:

- Bei einer **EREW–PRAM** (*exclusive read, exclusive write*) sind gleichzeitige Schreib– und Leseoperationen mehrerer Prozessoren verboten.
- Bei einer **CREW–PRAM** (*concurrent read, exclusive write*) sind nur gleichzeitige Leseoperationen mehrerer Prozessoren erlaubt.
- Bei einer **CRCW–PRAM** (*concurrent read, concurrent write*) sind sowohl gleichzeitige Schreib– als auch gleichzeitige Leseoperationen mehrerer Prozessoren erlaubt. Zur Regelung, welcher Prozessor sich bei einem Schreibkonflikt durchsetzt, unterscheidet man weitere Subtypen (vgl. hierzu [Ja92]).

Viele Resultate über **PRAM**s findet man in dem Survey [Vis83].

Sei P ein algorithmisches Problem und $O(t_{\text{seq}})$ die Zeit, die ein schnellster sequentieller Algorithmus benötigt, um das Problem P zu lösen. Ein paralleler Algorithmus, der das Problem P in Zeit $O(t)$ mit $O(p)$ Prozessoren löst, heißt *speedup–optimal*, falls $O(p \cdot t) = O(t_{\text{seq}})$.

Jeden parallelen Algorithmus kann man *sequentialisieren*, indem man die Arbeit der Prozessoren taktweise nacheinander sequentiell ausführt. Ein solcher sequentieller Algorithmus benötigt dann die Zeit $O(p \cdot t)$. In dieser Hinsicht liefert ein speedup–optimaler paralleler Algorithmus einen schnellsten sequentiellen Algorithmus.

Seien $J_1, \ldots, J_n$ (unabhängige) Jobs (ein Job sei hierbei eine Folge von Grundrechenoperationen, d.h. ein Programm auf einer **RAM**). Die sequentielle Ausführungszeit von J_i, $i = 1, \ldots, n$ betrage t_i. Gilt dann $t_i \in O(\log n)$ für alle $i = 1, \ldots, n$ sowie $\sum_{i=1}^{n} t_i \in O(n)$,[4] so lassen sich die Jobs $J_1, \ldots, J_n$ auf einer **EREW–PRAM** in $O(\log n)$ Zeit unter Benutzung von $O(n/\log n)$ Prozessoren ausführen ([CV86a]).

Läßt sich P parallel in Zeit $O(\log n)$ mit $O(n/\log n)$ Prozessoren lösen, wobei n die Länge der Eingabe bezeichne, so sprechen wir von einem *optimalen* parallelen Algorithmus bzw. von einem *optimal* parallel lösbarem Problem. Damit läßt sich jedes optimal parallel lösbare Problem sequentiell in Linearzeit lösen.

Nun zu einigen Grundproblemen, die für viele parallele Algorithmen benötigt werden.

Es sei $L = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n$ eine lineare Liste und $f : \{v_1, \ldots, v_n\} \rightarrow \mathbb{N}$ eine Bewertung der Listenelemente. Weiter sei $\otimes : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ eine assoziative binäre Verknüpfung auf $\mathbb{N}$ derart, daß sich für alle $i, j \in \mathbb{N}$ der Funktionswert $i \otimes j := \otimes(i, j)$ sequentiell in konstanter Zeit $O(1)$ berechnen läßt. Das *Partialsummenproblem (partial sum computation problem, prefix computation problem)* besteht nun darin, für jedes $k \in \{1, \ldots, n\}$ den Wert $d_f(v_k) := \bigotimes_{i=1}^{k} f(v_i)$ zu bestimmen.

Satz 1.2.1 ([CV86b])
Das Partialsummenproblem läßt sich auf einer **EREW–PRAM** *optimal parallel lösen.* ■

Folgende Spezialfälle des Partialsummenproblems treten häufig auf:

- *Gewichtetes list ranking Problem.*
  Hier ist speziell $\otimes := +$.

- *List ranking Problem.*
  Ist der Spezialfall des gewichteten list ranking Problems mit

$$f(v) := \begin{cases} 0, & \text{falls } v = v_1, \\ 1, & \text{falls } v \neq v_1. \end{cases}$$

  $d_f(v)$ gibt dann den Abstand von v zum Listenanfang an, den man auch als *Rang* $\mathrm{rank}(v)$ von v in der Liste L bezeichnet.

- *Maximums– bzw. Minimumsberechnung.*
  Um das Maximum bzw. Minimum der Elemente einer linearen Liste zu berechnen, setzt man $\otimes := \max$ bzw. $\otimes := \min$. $d_f(v_n)$ ist dann das Maximum bzw. Minimum der Listenelemente.

In parallelen Algorithmen für Bäume ist es meist nötig, daß der Baum als Wurzelbaum vorliegt. Sei $T = (V, E)$ ein Baum und $w \in V$. Dann lassen sich die unmittelbaren Vorgänger und unmittelbaren Nachfolger der Knoten aus T_w optimal parallel berechnen ([GR88], p. 22f). Im folgenden verwenden wir die Funktion $\mathrm{Root{-}Tree}(T, w)$, die den Baum T in den Wurzelbaum T_w „konvertiert“.

[4] Dies ist insbesondere dann der Fall, wenn $t_1 = \ldots = t_n = 1$ gilt.

1.2.4 Grundlagen aus der Komplexitätstheorie

Eine ausführliche Darstellung dieses Themengebiets findet man in [GJ79] und [Rei90].

Da man jedes Entscheidungsproblem, entsprechend codiert, als Akzeptierungsproblem einer Sprache beschreiben kann, beschränken wir uns im folgenden auf Akzeptierungsprobleme.

Dazu sei Σ ein Alphabet und Σ^* die Menge der Wörter über Σ.

Dann besteht $\mathbb{P}$ (bzw. $\mathbb{NP}$) aus allen Sprachen $L \subseteq \Sigma^*$, für die eine deterministische (bzw. nichtdeterministische) TURING Maschine existiert, die L in Polynomialzeit akzeptiert.

Eine Funktion f auf Σ^* heißt *polynomialzeit–berechenbar*, wenn sie sich auf einer deterministischen TURING–Maschine in polynomialer Zeit berechnen läßt.

Sind A, B Sprachen über Σ, so schreiben wir $A \leq_{\text{pol}} B$ (*A ist polynomialzeit–reduzierbar auf B*), wenn es eine polynomialzeit–berechenbare Funktion f auf Σ^* gibt, so daß für alle $x \in \Sigma^*$ gilt $x \in A \Longleftrightarrow f(x) \in B$.

Eine Sprache A über Σ heißt $\mathbb{NP}$–vollständig, wenn

- $A \in \mathbb{NP}$.
- Für alle $B \in \mathbb{NP}$ gilt $B \leq_{\text{pol}} A$.

Offensichtlich gilt $\mathbb{P} \subseteq \mathbb{NP}$, doch die Frage, ob $\mathbb{P} \neq \mathbb{NP}$ gilt, ist schon seit langer Zeit ein ungeklärtes Problem der Komplexitätstheorie. Jedoch geht man davon aus, daß $\mathbb{P} \neq \mathbb{NP}$ gilt. Diese Vermutung wird zudem dadurch bekräftigt, daß bis heute eine Vielzahl $\mathbb{NP}$–vollständiger Probleme bekannt sind, für die man keine polynomialen Algorithmen kennt.

Um die $\mathbb{NP}$–Vollständigkeit einer Sprache zu beweisen, nutzt man bekannte, schon als $\mathbb{NP}$–vollständig bewiesene, Sprachen aus, denn es folgt aus der Transitivität der Relation „$\leq_{\text{pol}}$“:

Satz 1.2.2
Es seien A, B Sprachen über Σ und $B \in \mathbb{NP}$. Dann gilt: Ist A $\mathbb{NP}$–vollständig und gilt $A \leq_{\text{pol}} B$, so ist auch B $\mathbb{NP}$–vollständig.

Analog läßt sich eine Theorie entwickelt, um die Parallelisierbarkeit von Problemen zu untersuchen.

Die Klasse $\mathbb{NC}$ (*Nick's Class*)[5] umfaßt genau diejenigen Sprachen, die sich auf einer **PRAM** in polylogarithmischer Zeit unter Benutzung von polynomial vielen Prozessoren akzeptieren lassen.[6]

Die Inklusion $\mathbb{NC} \subseteq \mathbb{P}$ ist klar, doch ob die Inklusion echt ist, ist ungeklärt. (Analogon zum Problem $\mathbb{P} \neq \mathbb{NP}$.)

Wie bei den $\mathbb{NP}$–vollständigen Problemen gibt es auch hier Probleme, die schwer parallelisierbar zu sein scheinen, die sogenannten $\mathbb{P}$–vollständigen Probleme.

Eine Funktion f auf Σ^* heißt *logspace–berechenbar*, wenn es eine deterministische TURING–Maschine M mit Eingabe–, Arbeits– und Ausgabeband gibt, die f auf logarithmischen Raum berechnet.

[5] Der Name geht zurück auf Nicholas Pippenger.

[6] Der Typ der **PRAM** spielt hier keine Rolle, da man die verschiedenen Typen untereinander in logarithmischer Zeit simulieren kann, vgl. hierzu das Survey [Har94].

Wir schreiben $A \leq_{\text{logspace}} B$ (*A ist logspace–reduzierbar auf B*) genau dann, wenn eine logspace–berechenbare Funktion f auf Σ^* existiert, so daß für alle $x \in \Sigma^*$ gilt $x \in A \iff f(x) \in B$.

Ein Sprache A über Σ heißt $\mathbb{P}$*–vollständig*, wenn gilt:

- $A \in \mathbb{P}$.
- Für jedes $B \in \mathbb{P}$ gilt $B \leq_{\text{logspace}} A$.

Recht viele Probleme, auch wenn sie sich sequentiell in Linearzeit lösen lassen, sind $\mathbb{P}$–vollständig. Eines davon ist z.B. die Bestimmung einer **DFS**–Numerierung (= Numerierung der Knoten gemäß Tiefensuche) eines Graphen. Weitere $\mathbb{P}$–vollständige Probleme findet man z.B. in [GR88] oder [Ja92].

Kapitel 2

Auswertung algebraischer Ausdrücke

Im ersten Abschnitt dieses Kapitels zeigen wir zunächst, wie man einen Wurzelbaum T in einen regulären binären Wurzelbaum $B(T)$ konvertiert.

Will man ein Problem parallel auf einem Wurzelbaum T lösen, so ist dies meist problematisch, falls T einen „langen“ Pfad P enthält, dessen innere Knoten zu keinem Knoten aus $T - P$ adjazent sind. Dieses Problem läßt sich meist dadurch umgehen, indem man zunächst das Problem auf $B(T)$ löst und dann hieraus eine Lösung für T erhält.

Im zweiten Abschnitt geben wir den optimalen Baum–Kontraktionsalgorithmus von ABRAHAMSON ET ALTERA aus [ADKP89] an. Dieser wird zur Auswertung algebraischer Ausdrücke benützt.

Es gibt zwei verschiedene Methoden, um algebraische Ausdrücke auf Bäumen auszuwerten, ein Bottom–up und ein Top–down Verfahren, diese beschreiben wir im dritten Abschnitt.

2.1 Konvertierung eines Wurzelbaums in einen regulären Wurzelbaum

Es sei $T_w = (V, E)$ ein Wurzelbaum. T_w heißt ein *binärer Wurzelbaum* (kurz: *Binärbaum*), wenn jeder Knoten höchstens zwei Kinder besitzt. Ein Graph G heißt *regulär*, falls in G alle Knoten denselben Grad besitzen. Einen Wurzelbaum T_w nennen wir *regulär*, wenn jeder innere Knoten (d.h. ein Knoten, der kein Blatt ist) aus T genau zwei Kinder besitzt, die man (der Reihenfolge in der linearen Liste nach) *linkes* bzw. *rechtes Kind* nennt. Jeder reguläre Wurzelbaum ist natürlich ein Binärbaum.

Es sei $T_w = (V, E)$ ein Wurzelbaum. Dann konvertieren wir T wie folgt in einen regulären Wurzelbaum $B(T) = (V', E')$:

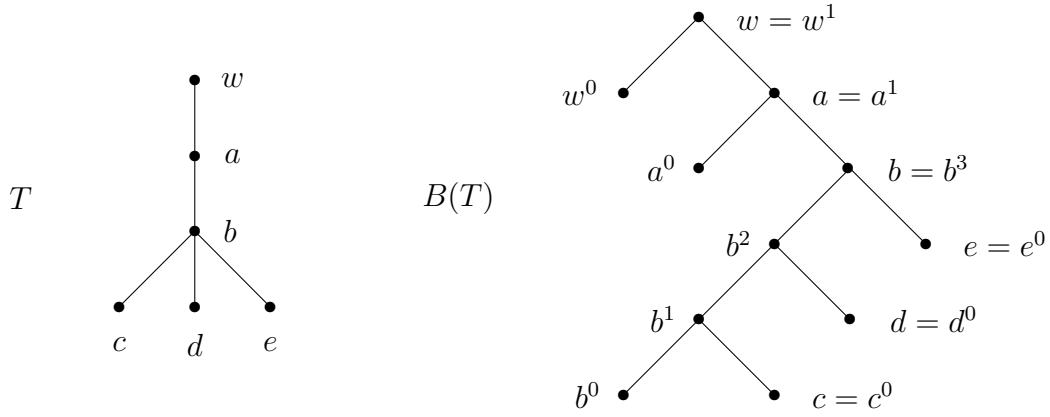
(1) (Wurzel)

Die Wurzel von $B(T)$ ist w.

(2) (Knoten)

Für jeden Knoten $v \in V$ mit Ausgangsgrad $l = \mathrm{outdeg}(v)$ nehmen wir Knoten $v^0, \ldots, v^{l-1}, v^l = v$ in V' auf.

Abbildung 2.1: Beispiel einer Konvertierung von T in $B(T)$



(3) (Kanten)

Die Kantenmenge E' besteht aus zwei Komponenten:

(a) (interne Kanten E'_{int})
Für jeden Knoten $v \in V$ sei für $i = 0, \ldots, \text{outdeg}(v) - 1$ der Knoten v^i das linke Kind von v^{i+1}.

(b) (externe Kanten E'_{ext})
Sei v ein Knoten aus V mit Kindern $u_1, \ldots, u_k$. Die Numerierung der Kinder sei dabei durch die Position in der linearen Liste der Kinder von v gegeben. In $B(T)$ sei dann u_i das rechte Kind von v^i für $i = 1, \ldots, \text{outdeg}(v)$.

Ein Beispiel für diese Konvertierung ist in Abbildung 2.1 dargestellt.

Ist $L : E \to \mathbb{N}$ eine Kantenbewertung eines Graphen $G = (V, E)$ und sind $u, v \in V$, so sei

$$L(u, v) := \min \left\{ \sum_{i=1}^{k-1} L(w_i, w_{i+1}) : u = w_1 - w_2 - \ldots - w_k = v \right\}.$$

Lemma 2.1.1
Es sei $T_w = (V, E)$ ein Wurzelbaum und $B(T) = (V', E')$ wie oben definiert. Dann gelten folgende Aussagen:

(1) Ist $|V| > 1$ so gilt:

(a) $B(T)$ ist ein regulärer Wurzelbaum mit Wurzel w.

(b) Die Menge aller Blätter in $B(T)$ ist gegeben durch $\{v^0 : v \in V\}$.

(2) Die internen Kanten von $B(T)$ sind genau die Linkskanten in $B(T)$ (eine Kante $e = \{u, v\}$, $u = \text{parent}(v)$, heißt eine Linkskante*, wenn v das linke Kind von u ist), und es gilt $|E'_{\text{ext}}| = |E|$.*

(3) $|V'| = 2|V| - 1$, $|E'| = 2|E|$.

(4) Sei $L(e) = 1$ für alle externen Kanten und $L(e) = 0$ für alle internen Kanten. Dann gibt für alle $u, v \in V$ der Wert $L(u, v)$ in $B(T)$ den Abstand $d_T(u, v)$ von u und v in T an.

(5) Die **DFS***–Reihenfolge (= Numerierung der Knoten gemäß dem Tiefensuche–Algorithmus) von $B(T)$ mit Startknoten w induziert in natürlicher Weise die* **DFS***–Reihenfolge von T mit Startknoten w.*[1]

(6) (Komplexität der Berechnung von $B(T)$)

(a) $B(T)$ kann sequentiell in Linearzeit konstruiert werden.

(b) $B(T)$ kann auf einer **EREW-PRAM** *optimal parallel konstruiert werden.*

Beweis. Die Aussagen (1) und (2) folgen direkt aus der Konstruktion von $B(T)$. Da in einem Baum $T = (V, E)$ gilt $|E| = |V| - 1$, folgt

$$|V'| = \sum_{v \in V} (1 + \mathrm{outdeg}(v)) = 1 + \sum_{v \in V} \deg(v) = 1 + 2|E| = 2|V| - 1$$

und $|E'| = |V'| - 1 = 2|E|$.

(4) und (5) folgen induktiv aus der Konstruktion von $B(T)$.

Nun zur Berechnung von $B(T)$. Zunächst numerieren wir die Kinder jedes Knotens gemäß ihrer Reihenfolge in der linearen Liste der Kinder (list ranking Problem). Nach Satz 1.2.1 geht dies optimal.[2] Der Wert $\mathrm{nr}(v)$ gebe für $v \in V \setminus \{w\}$ die Position von v in der Adjazenzliste von $\mathrm{parent}(v)$ an.

Da nach (3) gilt $|V'| = 2|V| - 1$ und wir die Ausgangsgrade der Knoten aus T bereits kennen ($\mathrm{outdeg}(v)$ = Rang des letzten Knotens in der Adjazenzliste von v), können wir die lineare Liste, welche die Knoten aus V' enthält, optimal erzeugen.

Da $B(T)$ ein regulärer Wurzelbaum ist, benutzen wir für jeden Knoten $v \in V'$ anstatt der linearen Liste der Kinder von v zwei Parameter $\mathrm{Lchild}(v)$ bzw. $\mathrm{Rchild}(v)$, die das linke bzw. rechte Kind von v in $B(T)$ enthalten (falls v ein innerer Knoten ist). Die Kanten von $B(T)$ können dann wie folgt optimal erzeugt werden: Ist $v \in V$ ein Knoten mit $v \neq w$, $i := \mathrm{nr}(v)$ und $u := \mathrm{parent}(v)$, so setzen wir

$$\mathrm{Lchild}(u_i) := u_{i-1}, \qquad \mathrm{Rchild}(u_i) := v. \qquad \blacksquare$$

2.2 Der Baum–Kontraktionsalgorithmus von Abrahamson et altera

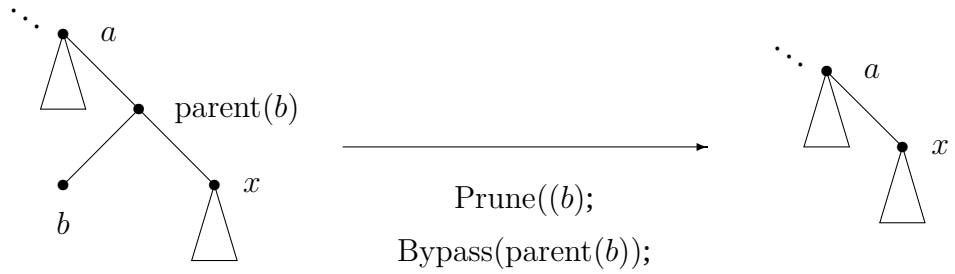
Sei T_w ein binärer Wurzelbaum. Der Algorithmus nutzt zwei fundamentale Operationen aus:

(P) $\mathrm{Prune}(b)$: Ist b ein Blatt in T, so entfernt die Operation $\mathrm{Prune}(b)$ den Knoten b aus T.

[1] Wenn wir von *der* **DFS**–Reihenfolge sprechen, so gehen wir davon aus, daß die Knoten im **DFS**–Algorithmus gemäß ihrer Reihenfolge in der Adjazenzliste durchlaufen werden.

[2] Wir fassen die einzelnen Adjazenzlisten als eine lineare Liste L auf. Die Länge von L liegt dann in $O(2|E|)$. Dann lösen wir das list ranking Problem auf L und bestimmen anschließend hieraus den Rang der Knoten für die einzelnen Adjazenzlisten.

Abbildung 2.2: Anwendung der Operationen $\mathrm{Prune}(b)$, $\mathrm{Bypass}(\mathrm{parent}(b))$ auf ein Blatt b



(B) $\mathrm{Bypass}(v)$: Es sei $v \neq w$ ein Knoten aus T, der genau einen unmittelbaren Nachfolger u besitzt. Die Operation $\mathrm{Bypass}(v)$ entfernt v aus T und setzt $\mathrm{parent}(u) := \mathrm{parent}(v)$.

Es seien T, T' zwei binäre Wurzelbäume. Weiter seien $P := \{b \in V(T) : b \text{ Blatt in } T\}$, $B := \{v \in V(T) : v \neq w \text{ besitzt genau einen Sohn}\}$ sowie $W := V(T) \setminus V(T') \subseteq B \cup P$. T' heißt eine *(Baum–)Kontraktion* von T, wenn T' wie folgt aus T hervorgeht: Zunächst wird, ausgehend von T, auf die Knoten aus $W \cap B$ die Operation (B) angewendet. Anschließend wendet man auf die Knoten aus $W \cap P$ die Operation (P) an.

Um die parallele Ausführbarkeit der einzelnen Operationen einer Kontraktion zu sichern, benötigen wir spezielle Kontraktionen. Es sei T' eine Kontraktion von T und $W := V(T) \setminus V(T')$. Gilt dann $\mathrm{parent}(v) \notin W$ für alle $v \in W$, so heißt T' eine *unabhängige Kontraktion* von T.

Eine Folge $T_1, \ldots, T_k$ von Bäumen heißt eine *(unabhängige) (Baum–)Kontraktionsfolge* von T, wenn sie folgende Eigenschaften erfüllt:

(1) $T_1 = T$,

(2) $T_k = (\{w\}, \emptyset)$,

(3) Für alle $i \in \{1, \ldots, k-1\}$ ist T_{i+1} eine (unabhängige) Kontraktion von T_i.

Die Zahl k nennen wir die *Länge* der Kontraktionsfolge.

Natürlich besitzt jeder Wurzelbaum T_w eine Kontraktionsfolge der Länge $\mathrm{depth}(T)$, indem man im i–ten Schritt, $i = 1, \ldots, \mathrm{depth}(T)$, auf alle Knoten v mit $\mathrm{depth}(v) = \mathrm{depth}(T) - i + 1$ die Operation (P) anwendet. Für optimale Algorithmen benötigt man aber Kontraktionsfolgen der Länge $O(\log |V|)$. Dazu definieren wir: Eine Kontraktionsfolge der Länge k eines Wurzelbaumes $T = (V, E)$ heißt *optimal*, wenn $k \in O(\log |V|)$.

In [ADKP89] wird gezeigt, daß jeder binäre Wurzelbaum eine optimale Baum–Kontraktionsfolge besitzt und daß sie effizient berechnet werden kann. Hierbei kann man sich auf reguläre Wurzelbäume beschränken, da sich aus einer Kontraktionsfolge von $B(T)$ leicht eine Kontraktionsfolge von T berechnen läßt.

Die Grundidee des Algorithmus besteht darin, in jedem Schritt parallel auf bestimmte Blätter v in T nacheinander die Operationen $\mathrm{Prune}(v)$, $\mathrm{Bypass}(\mathrm{parent}(v))$ auszuführen, vgl. Abbildung 2.2. Hierbei muß natürlich die Unabhängigkeit der Kontraktionen gewährleistet bleiben. Dies geschieht dadurch, daß in jedem Schritt k folgendes beachtet wird:

(1) Auf ein Blatt v (in T) mit Nummer $\mathrm{nr}(v)$ (aus Zeile (2)) wird im Schritt k genau dann die Operation Prune angewendet, wenn

$$k = \min\{j \in \{1, \ldots, \log|V|\} : \mathrm{nr}(v) \bmod 2^{j-1} = 1\},$$

d.h. in der Darstellung von $\mathrm{nr}(v)$ als Binärzahl ist k die erste Stelle von rechts, wo eine 1 steht.

(2) Die sequentielle Abarbeitung der Zeilen (9) bis (18) sichert, daß nicht auf zwei benachbarte Knoten die Operation Bypass gleichzeitig angewendet wird (d.h. in jedem Schritt werden (nacheinander) zwei Kontraktionen durchgeführt).

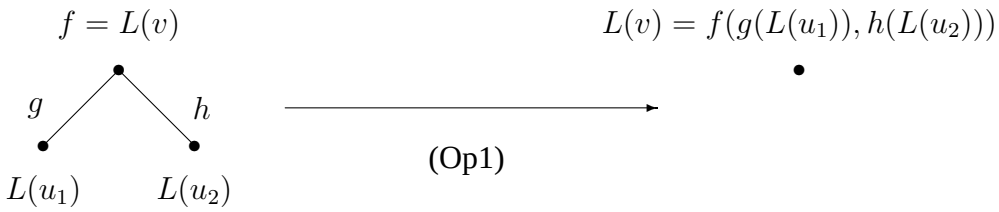
Algorithmus von Abrahamson, Dadoun, Kirkpatrick und Przytycka
Eingabe : Ein regulärer binärer Wurzelbaum $T = (V, E)$ mit Wurzel w.
Ausgabe : Ein optimale Kontraktionsfolge von T.[3]

```
(1)  begin
(2)    Bestimme bei 0 beginnend parallel die Numerierung nr der Blätter von T
       von links nach rechts;
(3)    repeat log|V| times
(4)      for each Blatt b in parallel do
(5)        begin
(6)          v := parent(b);
(7)          if (nr(b) ist ungerade) and (v ≠ w)
(8)            then begin
(9)                   if b ist linkes Kind von v
(10)                    then begin
(11)                           Prune(b);
(12)                           Bypass(v);
(13)                         end
(14)                  if b ist rechtes Kind von v
(15)                    then begin
(16)                           Prune(b);
(17)                           Bypass(v);
(18)                         end
(19)                end
(20)           else if v ≠ w
(21)                  then nr(b) := nr(b)/2
(22)                  else Prune(b);
(23)       end
(24)  end.
```

Satz 2.2.1 ([ADKP89])
Der obige Algorithmus ist korrekt und läßt sich auf einer **EREW-PRAM** *in $O(\log|V|)$ Zeit mit $O(|V|/\log|V|)$ Prozessoren implementieren.* ■

[3] Hierbei wird nicht die gesamte Kontraktionsfolge erzeugt, sondern es werden nur die Knoten angegeben, die in jedem Kontraktionsschritt entfernt werden.

Abbildung 2.3: Die Operation (Op1)



2.3 Die beiden Verfahren zur Auswertung algebraischer Ausdrücke

Sei T_w ein regulärer Wurzelbaum. Weiter sei S eine Menge, $F \subseteq \{f : f : S \times S \to S\}$ eine Teilmenge von binären Funktionen auf S und $H \subseteq \{h : h : S \to S\}$ eine Teilmenge von unären Funktionen auf S.

Es gibt zwei verschiedene Methoden zur Auswertung von algebraischen Ausdrücken (**ATC** = *algebraic tree computation*) auf Bäumen, ein Bottom–up und ein Top–down Verfahren, die wir im folgenden näher beschreiben werden. In beiden Verfahren ist eine Belegungsfunktion $L : V \uplus E \to S \cup F \cup H$ gegeben.

2.3.1 Das Bottom–up Verfahren B–ATC

Hier sind die Blätter mit Elementen aus S, die inneren Knoten mit Elementen aus F und die Kanten mit Elementen aus H belegt. Das **B–ATC** (*Bottom–up algebraic tree computation*) Problem besteht nun darin, die Endbelegung der Wurzel zu bestimmen.

Wir betrachten die folgende Operation (Op1), vgl. auch Abbildung 2.3:

(Op1) Ist v ein Knoten mit $L(v) = f$, dessen Kinder u_1, u_2 Blätter sind, so entfernen wir u_1, u_2 aus T und aktualisieren $L(v)$ durch $f(g(L(u_1)), h(L(u_2)))$, wobei $L(vu_1) = g$ und $L(vu_2) = h$.

Da T nach Voraussetzung ein regulärer Wurzelbaum ist, läßt sich T durch wiederholte Anwendung von (Op1) zur Wurzel reduzieren. Die Belegung der Knoten aus T nach dieser Reduktion (mit Belegung sei hier der L–Wert gemeint, wenn nur noch die Wurzel übrig ist) nennen wir die *Endbelegung* der Knoten.

Damit das **B–ATC** Problem effizient lösbar ist, muß man natürlich einige zusätzliche Forderungen an die beteiligten Funktionen stellen. Ein **B–ATC** Problem heißt *zerlegbar* genau dann, wenn die folgenden beiden Eigenschaften erfüllt sind:

(B1) Die Mengen F und H seien als indizierte Mengen gegeben und die Funktionswerte der Funktionen aus F und H können in $O(1)$ sequentieller Zeit berechnet werden.

(B2) Für alle $h_i, h_j, h_k \in H$, $f_l \in F$ und $a \in S$ gehören die Funktionen $x \mapsto h_i(f_l(h_j(x), h_k(a)))$ und $x \mapsto h_i(f_l(h_k(a), h_j(x)))$ zu H und ihre Indizes können in $O(1)$ sequentieller Zeit berechnet werden.

Dann gilt:

Satz 2.3.1 ([ADKP89])
Zerlegbare **B–ATC** *Probleme lassen sich auf einer* **EREW–PRAM** *in Zeit* $O(\log n)$ *mit* $O(n/\log n)$ *Prozessoren lösen. Ferner kann die Endbelegung aller Knoten des Baumes in derselben Zeit– und Prozessorkomplexität berechnet werden.* ■

Grundlage des Beweises von Satz 2.3.1 ist der optimale parallele Baum–Kontraktionsalgorithmus aus dem zweiten Abschnitt.

Aus Satz 2.3.1 folgt sofort, daß sich jedes zerlegbare **B–ATC** Problem sequentiell in Linearzeit lösen läßt, indem man den parallelen Algorithmus sequentialisiert. Eine andere Möglichkeit, um sequentiell die Endbelegung der Wurzel (oder aller Knoten) in Linearzeit zu bestimmen, besteht darin, den Baum rekursiv mit postorder (vgl. [Bra94], Seite 23f) auszuwerten. Hierzu verwendet man die folgende Funktion:

function Endbelegung(v : KnotenTyp);
begin
  if v ist ein Blatt
    then Endbelegung(v) := $L(v)$;
    else begin
        { v ist ein innerer Knoten von T }
        Sei u_L das linke Kind und u_R das rechte Kind von v;
        a := Enbelegung(u_L);
        b := Enbelegung(u_R);
        Enbelegung(v) := $L(v)(a, b)$;
      end
end.

Der Wert Endbelegung(w) gibt dann die Endbelegung der Wurzel an.

2.3.2 Das Top–down Verfahren T–ATC

Hier wird die Wurzel mit einem Element aus S, die inneren Knoten und die Kanten werden mit einem Element aus H belegt. Das **T–ATC** (*Top–down algebraic tree computation*) Problem besteht darin, die Endbelegung aller Knoten aus T zu berechnen.

Wir betrachten die Operation (Op2):

(Op2) Ist v ein Knoten mit $L(v) \in S$ und u ein Kind von v mit $L(u) \notin S$, so aktualisieren wir $L(u)$ durch $h(g(L(v)))$, wobei $g = L(uv)$ und $h = L(u)$.

Durch wiederholte Ausführung von (Op2) erreicht man, daß alle Knoten aus T mit Werten aus S belegt sind. Diese Belegung nennen wir dann die *Endbelegung* der Knoten.

Analog zum **B–ATC** Problem braucht man auch für das **T–ATC** Problem gewisse zusätzliche Voraussetzungen, damit das Problem effizient lösbar wird. Ein **T–ATC** Problem heißt *zerlegbar* genau dann, wenn die folgenden beiden Eigenschaften erfüllt sind:

(T1) Die Menge H ist indiziert und die Funktionswerte der Funktionen aus H können in $O(1)$ sequentieller Zeit berechnet werden.

(T2) Für alle $h_i, h_j \in H$ gehört die Funktion $h_i \circ h_j$ zu H und ihr Index kann in $O(1)$ sequentieller Zeit berechnet werden.

Dann gilt:

Satz 2.3.2 ([ADKP89])
Zerlegbare **T–ATC** *Probleme lassen sich auf einer* **EREW–PRAM** *in Zeit* $O(\log n)$ *mit* $O(n/\log n)$ *Prozessoren lösen.* ■

2.3.3 Die Funktionen B–ATC und T–ATC

Sei T_w ein (nicht notwendig regulärer und binärer) Wurzelbaum, $l : V \to S$ eine Belegung der Knoten aus T mit Werten aus S, $a \in S$ und $f \in F$ sowie $h_e \in H$ für alle $e \in E$.

Im folgenden benutzen wir häufig die zwei Funktionen

$$\textbf{B–ATC}(T_w, l, f) \quad \text{und} \quad \textbf{T–ATC}(T_w, a, (h_e)_{e\in E}),$$

die wie folgt arbeiten: Zunächst konvertieren beide Funktionen den Baum T_w in den regulären Wurzelbaum $T' := B(T_w)$.

Die Funktion **B–ATC**(T_w, l, f) löst anschließend das **B–ATC** Problem mit der Belegung L auf T', wobei

- $L(v^0) := l(v)$ für alle Knoten v aus T (man beachte, daß nach Lemma 2.1.1 die Menge der Blätter in T' durch $\{v^0 : v \in V\}$ gegeben ist),
- $L(v) := f$ für alle inneren Knoten v aus T',
- $L(e) := \mathrm{id}$ für alle Kanten aus T', wobei id die Identitätsfunktion auf S bezeichne.

Für jeden Knoten v aus T liefert die Funktion einen Wert aus S zurück und zwar die Endbelegung von v bzgl. des eben angegebenen **B–ATC** Problems auf T'.

Analog löst **T–ATC**$(T_w, a, (h_e)_{e\in E})$ das **T–ATC** Problem auf T' mit der folgenden Initialisierung

- $L(w) = a$ für die Wurzel w,
- $L(v) = \mathrm{id}$ für alle Knoten $v \neq w$ aus T',
- $L(e) = \mathrm{id}$ für alle internen Kanten und $L(e) = h_{e'}$ für jede externe Kanten $e = u_i v_j$, wobei $e' = uv$.

Sind alle Funktionen h_e, $e \in E$, identisch, so schreiben wir für **T–ATC**$(T_w, a, (h_e)_{e\in E})$ kurz **T–ATC**(T_w, a, h), wobei $h = h_e$.

Aus Lemma 2.1.1 folgt sofort

Lemma 2.3.3
Sei $T_w = (V, E)$ *ein Wurzelbaum und* $g_1 :=$ **T–ATC**$(T_w, a, (h_e)_{e\in E})$*. Weiter gebe* g_2 *die Endbelegung der Knoten aus* T *an, wenn man das* **T–ATC** *Problem mit der Initialisierung*

$$L(x) = \begin{cases} \mathrm{id}, & \textit{falls } x \in V, \\ h_x, & \textit{falls } x \in E \end{cases}$$

auf T_w löst. Dann gilt $g_1 = g_2$. ■

Viele Graphenklassen lassen sich durch Eliminationsordnungen charakterisieren. So auch die Klasse der Bäume. Eine Folge $\sigma = (v_1, \ldots, v_n)$ von paarweise verschiedenen Knoten aus T heißt eine *Blätter–Eliminationsordnung* von T, wenn $V = \{v_1, \ldots, v_n\}$ und für $i = 1, \ldots, n-1$ der Knoten v_i ein Blatt in $G_i := G(\{v_i, \ldots, v_n\})$ ist.

Entlang einer Blätter–Eliminationsordnung lassen sich viele Graphenprobleme mit folgender Strategie lösen: Zu Beginn werden alle Knoten des Baumes mit einem Wert initialisiert. Anschließend wird wiederholt ein Blatt b gelöscht und der Wert des Nachbarn von b nach einer bestimmten Regel aktualisiert, bis nur noch ein Knoten übrig bleibt. Aus den Endbelegungen der Knoten bestimmt man dann eine Lösung für das gegebene Problem.

Dieses sequentielle Verfahren läßt sich in eine Instanz eines **B–ATC** Problems umwandeln, wie im nächsten Lemma beschrieben ist:

Lemma 2.3.4
*Sei $T_w = (V, E)$ ein Wurzelbaum, $r : V \to M$ und $f : M \times M \to M$ zwei Funktionen. Weiter sei $L :=$ **B–ATC**(T, r, f). Dann existiert eine Blätter–Eliminationsordnung $\sigma = (v_1, \ldots, v_n)$ mit $v_n = w$, so daß wir L sequentiell wie folgt erhalten können: Wir initialisieren L durch r und anschließend löschen wir für $i = 1, \ldots, n-1$ nacheinander v_i und aktualisieren die Belegung des unmittelbaren Nachfolgers u_i von v_i in G_i durch $f(u_i, v_i)$.* ■

Kapitel 3

Effiziente parallele Algorithmen für Bäume

In diesem Kapitel geben wir effiziente parallele Algorithmen für Bäume an. In [HY88] wurden bereits optimale parallele Algorithmen für folgende Probleme (auf Bäumen) entwickelt:

- Berechnung einer minimalen Knotenüberdeckung (**Vertex Cover Problem**). Hierbei heißt eine Teilmenge $V' \subseteq V(G)$ eines Graphen G eine *Knotenüberdeckung*, wenn $e \cap V' \neq \emptyset$ für alle $e \in E(G)$ gilt.
- Berechnung einer maximalen unabhängigen Menge.
- Berechnung eines maximalen Matchings (**Maximum Matching Problem**). Ein Matching (oder *unabhängige Kantenmenge*) M in G ist eine Teilmenge von $E(G)$, so daß für alle $e_1, e_2 \in M$ mit $e_1 \neq e_2$ gilt $e_1 \cap e_2 = \emptyset$.

3.1 Abstandsberechnungen

Sei succ die Successor–Funktion auf $\mathbb{Z}$, d.h. $\mathrm{succ}(a) := a + 1$ für alle $a \in \mathbb{Z}$.

Lemma 3.1.1
*Sei T_w ein Wurzelbaum. Dann berechnet $L :=$ **T–ATC**$(T_w, 0, \mathrm{succ})$ für jeden Knoten v aus T den Abstand $d(v, w)$ von v zur Wurzel w, d.h. $L = \mathrm{depth}$.*

Beweis. Die Behauptung folgt unmittelbar aus Lemma 2.1.1 und Induktion. ■

Wir betrachten für einen Graphen G die folgenden zwei Abstandsprobleme:

- (**SSSP — single source shortest path**)
  Man möchte die Abstände aller Knoten aus G zu einem festen Knoten berechnen.
- (**APSP — all pairs shortest path**)
  Gesucht ist die Abstandsmatrix $(d_G(u, v))_{(u,v) \in V \times V}$.

Sequentiell läßt sich **SSSP** für beliebige Graphen mit **BFS** in Linearzeit lösen. Wir zeigen im folgenden, daß für Bäume beide Probleme optimal lösbar sind (bei **APSP** bezogen auf die Ausgabe).

Folgerung 3.1.2
Die Probleme **SSSP** *und* **APSP** *lassen sich auf Bäumen optimal parallel (bei* **APSP** *bezogen auf die Ausgabe) auf einer* **EREW–PRAM** *lösen.*

Beweis. Da das **T–ATC** Problem aus Lemma 3.1.1 zerlegbar ist, folgt die Behauptung unmittelbar aus Satz 2.3.2. ■

Das **SSSP** Problem läßt sich zum Problem **DistSet** wie folgt verallgemeinern: Sei $D \subseteq V$ eine Teilmenge von V. Dann ist für alle Knoten $v \in V$ der Abstand $d(v, D) = \min_{d \in D} d(v, d)$ von v zur Menge D gesucht. Wir zeigen, daß sich auch dieses Problem auf Bäumen optimal lösen läßt. Sei dazu die Funktion $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ definiert durch $f(a, b) := \min(a, b + 1)$.

Algorithmus PDistSet
Eingabe : Ein Baum $T = (V, E)$ und $D \subseteq V$.
Ausgabe : Eine Abbildung $g : V \to \mathbb{N}$ mit $g(v) = d(v, D)$ für alle $v \in V$.

(1) **begin**
(2) Sei w ein beliebiger Knoten aus T;
(3) $T_w := \mathrm{Root{-}Tree}(T, w)$;
(4) $n := |V|$;
(5) **for all** $v \in V$ **in parallel do**
(6) **if** $v \in D$
(7) **then** $L(v) := 0$
(8) **else** $L(v) := n$;
(9) $M :=$ **B–ATC**(T_w, L, f);
(10) $g :=$ **T–ATC**$(T_w, M(w), (\min(\cdot + 1, M(v)))_{e=vv' \in E, v > v'})$;[1]
(11) **return**(g);
(12) **end.**

Lemma 3.1.3
Sei $G := \{g_i : g_i : \mathbb{Z} \to \mathbb{Z}, i \in I\}$, *wobei* $I := \mathbb{Z}^2 \cup \{\infty\}$ *und für* $x \in \mathbb{Z}$

$$g_i(x) := \begin{cases} \min(x + a, b), & \text{falls } i = (a, b) \in \mathbb{Z}^2, \\ x, & \text{falls } i = \infty. \end{cases}$$

Dann gelten folgende Aussagen:

(1) G *ist bzgl. der üblichen Komposition „∘“ von Funktionen abgeschlossen, d.h.* $g_i \circ g_j \in G$ *für alle* $i, j \in I$.

(2) Es sei $h : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ *definiert durch* $h(x, y) := \min(x + a, y + b)$, *wobei* $a, b \in \mathbb{Z}$. *Dann gehört auch die Funktion* $h \circ (g_i, g_j)$ *für alle* $i, j \in I$ *zur Menge* G.

(3) Die Indizes der verketteten Funktionen (bzgl. der Menge G*), die in (1) und (2) auftreten, können in konstanter Zeit berechnet werden.*

Beweis.

[1] $v > v'$ stehe hier abkürzend für $\mathrm{depth}(v) > \mathrm{depth}(v')$.

(1) Ist einer der beiden Indizes i, j gleich ∞, so ist dies trivial. Daher sei $i := (a, b)$, $j := (c, d) \in \mathbb{Z}^2$. Dann gilt

$$\begin{aligned} g_{(a,b)} \circ g_{(c,d)} &= \min(\min(\cdot + c, d) + a, b) \\ &= \min(\min(\cdot + a + c, a + d), b) \\ &= \min(\cdot + (a + c), \min(a + d, b)) \\ &= g_{(a+c,\min(a+d,b))}. \end{aligned}$$

(2) Wir beweisen dies nur für den Fall, daß $i := (c, d)$, $j := (e, f) \in \mathbb{Z}^2$ sind. Dann gilt:

$$\begin{aligned} h \circ (g_i, g_j) &= \min(\min(\cdot + c, d) + a, \min(\cdot + e, f) + b) \\ &= \min(\min(\cdot + a + c, a + d), \min(\cdot + b + e, b + f)) \\ &= \min(\cdot + \min(a + c, b + e), \min(a + d, b + f)) \\ &= g_{(\min(a+c,b+e),\min(a+d,b+f))}. \end{aligned}$$

(3) Trivial. ■

Satz 3.1.4
Der Algorithmus **PDistSet** *ist korrekt und läßt sich auf einer* **EREW–PRAM** *optimal parallel implementieren.*

Beweis.

Korrektheit: Induktiv folgt aus Lemma 2.3.4, daß nach Zeile (9) für jeden Knoten $v \in V$ gilt

$$M(v) = \begin{cases} d(V(T_v) \cap D, v), & \text{falls } V(T_v) \cap D \neq \emptyset, \\ n, & \text{falls } V(T_v) \cap D = \emptyset, \end{cases}$$

also insbesondere $M(w) = d(D, w)$. Da für jeden Knoten $v \neq w$, $u := \mathrm{parent}(v)$, gilt

$$d(v, D) = \begin{cases} \min(d(V(T_v) \cap D, v), d(u, D) + 1), & \text{falls } V(T_v) \cap D \neq \emptyset, \\ d(u, D) + 1, & \text{falls } V(T_v) \cap D = \emptyset, \end{cases}$$

ist klar, daß nach Ausführung von (10) die Funktion g gleich der Funktion $d(\cdot, D)$ ist.

Komplexität: Dazu brauchen wir nur die Zerlegbarkeit des **B–ATC** bzw. **T–ATC** Problems in Zeile (9) bzw. (10) bestätigen. Die Zerlegbarkeit folgt aber unmittelbar aus Lemma 3.1.3. ■

3.2 Berechnung des kleinsten gemeinsamen Vorgängers von Knoten in Bäumen

Es sei $T_w = (V, E)$ ein Wurzelbaum und $U \subseteq V$ eine nichtleere Teilmenge von V. Dann gibt es genau einen Knoten $\mathrm{kgV}(U)$, der folgende Eigenschaften erfüllt:

(1) Für jeden Knoten $u \in U$ ist $\mathrm{kgV}(U)$ ein Vorgänger von u, d.h. $\mathrm{kgV}(U) \in \bigcap_{u \in U} P(u, w)$.

(2) $\mathrm{depth}(\mathrm{kgV}(U))$ ist maximal unter allen Knoten, die Eigenschaft (1) erfüllen.

Sind $u, v \in V$, so gilt $P(u,w) \neq P(v,w)$ genau dann, wenn $u \neq v$ ist. Damit ist $\mathrm{kgV}(U)$ eindeutig bestimmt durch

$$P(\mathrm{kgV}(U), w) = \bigcap_{u \in U} P(u,w).$$

Der Knoten $\mathrm{kgV}(U)$ heißt der *kleinste gemeinsame Vorgänger* der Knoten aus U (in T).

Lemma 3.2.1
Sei $T_w = (V,E)$ ein Wurzelbaum und $U \subseteq V$ eine nichtleere Teilmenge von V. Ist $U' \subset U$ eine nichtleere echte Teilmenge von U, so gilt

$$\mathrm{kgV}(U) = \mathrm{kgV}(\{\mathrm{kgV}(U')\} \cup (U \setminus U')).$$

Beweis. Die Aussage folgt sofort aus

$$\bigcap_{u \in U} P(u,w) = \bigcap_{u \in U'} P(u,w) \cap \bigcap_{u \in U \setminus U'} P(u,w) = P(\mathrm{kgV}(U'), w) \cap \bigcap_{u \in U \setminus U'} P(u,w). \qquad \blacksquare$$

Die Berechnung des kleinsten gemeinsamen Vorgängers zweier Knoten wurde bereits vielfach untersucht und es sind optimale parallele Algorithmen hierfür bekannt.

In [SV88] gehen die Autoren in zwei Phasen vor: In der ersten Phase berechnen sie für jeden Knoten $v \in V$ eine natürliche Zahl $\mathrm{Inlabel}(v)$. Diese Phase läßt sich auf einer **EREW–PRAM** optimal implementieren. Anschließend können sie, in einer zweiten Phase, den kgV zweier Knoten in konstanter Zeit aus den $\mathrm{Inlabel}$–Werten berechnen. Somit lassen sich k solche kgV–Berechnungen auf einer **CREW–PRAM** in Zeit $O(\log n)$ mit $O((n+k)/\log n)$ Prozessoren durchführen.

Aus Lemma 3.2.1 folgt somit, daß sich der kgV einer nichtleeren Teilmenge von V auf einer **CREW–PRAM** optimal berechnen läßt. Doch der Beweis für die Korrektheit und Implementierung des Algorithmus aus [SV88] ist recht aufwendig und technisch.

Wir presentieren hier einen einfachen optimalen parallelen Algorithmus, der den kleinsten gemeinsamen Vorgänger einer Teilmenge auf einer **EREW–PRAM** berechnet. Wir benötigen hierzu folgendes

Lemma 3.2.2
Sei $T_w = (V,E)$ ein Wurzelbaum und $U \subseteq V$ eine nichtleere Teilmenge von V. Dann gelten folgende Aussagen:

(1) $U \subseteq V(T_{\mathrm{kgV}(U)})$.

(2) $\mathrm{depth}(\mathrm{kgV}(U)) \leq \mathrm{depth}(\mathrm{kgV}(U'))$ *für jedes* $U' \subseteq U$.

(3) Ist $U = \{u\}$, *so folgt* $\mathrm{kgV}(U) = u$.

(4) $\mathrm{depth}(\mathrm{kgV}(U)) \leq \mathrm{depth}(u)$ *für jedes* $u \in U$.

(5) Aus $\mathrm{kgV}(U) \notin U$ *folgt*

(a) $|U| \geq 2$.

(b) $\mathrm{outdeg}(\mathrm{kgV}(U)) \geq 2$.

(c) Es gibt $u_1, u_2 \in U$ *mit* $\mathrm{kgV}(\{u_1, u_2\}) = \mathrm{kgV}(U)$.

Beweis. Die Aussagen (1) bis (3) folgen unmittelbar aus der Definition des kleinsten gemeinsamen Vorgängers. (4) folgt aus (2) und (3). Nun zum Beweis von (5): (a) folgt aus (3). $\mathrm{outdeg}(\mathrm{kgV}(U)) = 0$ würde $|U| = 1$ implizieren. Wäre $\mathrm{outdeg}(\mathrm{kgV}(U)) = 1$, v der Sohn von $\mathrm{kgV}(U)$, so folgt $v \in \bigcap_{u \in U} P(u, w)$ mit $\mathrm{depth}(u) > \mathrm{depth}(\mathrm{kgV}(U))$, Widerspruch. Also ist $\mathrm{outdeg}(\mathrm{kgV}(U)) \geq 2$. Seien $v_1, \ldots, v_k$ die Kinder von $\mathrm{kgV}(U)$ und $V_i := V(T_{v_i})$, $i = 1, \ldots, k$. Dann ist $|V_i \cap U| \geq 1$ für mindestens zwei Indizes i_1, $i_2 \in \{1, \ldots, k\}$. Sind $u_j \in V_{i_j} \cap U$, $j = 1, 2$, so erfüllen u_1, u_2 die Aussage (c). ■

Sei $\mathrm{or} : \{0,1\} \times \{0,1\} \to \{0,1\}$ die BOOLEsche ODER–Funktion, d.h. $\mathrm{or}(a, b) = 1$ genau dann, wenn $a = 1$ oder $b = 1$ ist.

Algorithmus PKGV
Eingabe : Ein Baum $T = (V, E)$ und eine nichtleere Teilmenge $U \subseteq V$ von V.
Ausgabe : Der Knoten $\mathrm{kgV}(U)$.

(1) **begin**
(2) Sei w ein beliebiger Knoten aus T;
(3) $T_w := \mathrm{Root{-}Tree}(T, w)$;
(4) $\mathrm{depth} :=$ **T–ATC**$(T_w, 0, \mathrm{succ})$;
(5) **for all** Knoten v aus T_w **in parallel do**
(6) **if** $v \in U$
(7) **then** $L(v) := 1$
(8) **else** $L(v) := 0$;
(9) $M :=$ **B–ATC**(T_w, L, or);
(10) $W := U$;
(11) **for all** v mit $M(v) = 1$ **in parallel do**
(12) **if (**$\mathrm{outdeg}(v) > 1$**) and (**mind. zwei Kinder von v haben einen M–Wert gleich 1)
(13) **then** $W := W \cup \{v\}$;
(14) Bestimme $v_{\mathrm{kgV}} \in W$ mit $\mathrm{depth}(v_{\mathrm{kgV}}) = \min_{u \in W} \mathrm{depth}(u)$;
(15) **return(**v_{kgV}**)**;
(16) **end.**

Lemma 3.2.3
Seien T_w *und* U *wie in Lemma 3.2.2. Weiter sei* L *die Indikatorfunktion auf* U*, d.h.*

$$L(v) := \begin{cases} 1, & \textit{falls } v \in U, \\ 0, & \textit{falls } v \notin U. \end{cases}$$

Dann ist das **B–ATC** *Problem* $M :=$ **B–ATC**(T_w, L, or) *zerlegbar, und es gilt*

$$(*) \qquad \{v \in V : M(v) = 1\} = \bigcup_{u \in U} P(u, w).$$

Beweis. Die Gleichung $(*)$ folgt induktiv aus $\mathrm{or}(1, \cdot) = 1$. Die Eigenschaften (B1) und (B2) sind für das **B–ATC** Problem erfüllt, da $S = \{0, 1\}$, $F = \{\mathrm{or}\}$ und $H = \{\mathrm{id}, 1\}$. ■

Lemma 3.2.4
Sei $T_w = (V, E)$ ein Wurzelbaum und $f : V \to \{0, 1\}$ eine BOOLE*sche Funktion auf V. Dann läßt sich die Funktion $g : V \to \{0, 1\}$, definiert durch*

$$g(v) := \begin{cases} 1, & \textit{falls } \mathrm{outdeg}(v) > 1 \textit{ und mind. zwei Kinder von } v \\ & \textit{den } f\textit{–Wert gleich } 1 \textit{ haben}, \\ 0, & \textit{sonst}, \end{cases}$$

auf einer **EREW–PRAM** *in $O(\log |V|)$ Zeit mit $O(|V|/\log |V|)$ Prozessoren berechnen.*

Beweis. Für einen Knoten v bezeichnen wir die lineare Liste seiner Kinder mit $\mathrm{Klist}(v)$. Das erste Element von $\mathrm{Klist}(v)$ sei v^e und das letzte v^l. Um die Anzahl der Kinder von v zu berechnen, die einen f–Wert gleich 1 besitzen, lösen wir folgendes gewichtete list–ranking Problem auf $\mathrm{Klist}(v)$: Jeder Knoten u aus $\mathrm{Klist}(v)$ wird mit $f(u)$ bewertet. Das Ergebnis des gewichteten list–ranking Problems auf $\mathrm{Klist}(v)$ bezeichnen wir mit b_v. (Ist $\mathrm{Klist}(v)$ eine leere Liste, so soll $b_v = 0$ sein.) Dann ist $g(v) = 1$ genau dann, wenn $b_v \geq 2$ ist. Die angegebene Zeit– und Prozessorkomplexität erreicht man, da die Gesamtanzahl der Elemente in den Listen $\mathrm{Klist}(v)$, $v \in V$, in $O(n)$ liegt. ■

Korollar 3.2.5
Der Algorithmus **PKGV** *bestimmt den kleinsten gemeinsamen Vorgänger zu einer Knotenmenge auf einer* **EREW–PRAM** *in $O(\log |V|)$ Zeit mit $O(|V|/\log |V|)$ Prozessoren.*

Beweis. Wir nehmen zunächst an, daß $\mathrm{kgV}(U) \notin W$ wäre. Da nach Zeile (10) des Algorithmus $U \subseteq W$ ist, erhalten wir $\mathrm{kgV}(U) \notin U$. Nach Lemma 3.2.2 hat dies zur Folge, daß $\mathrm{outdeg}(\mathrm{kgV}(U)) > 1$ und mindestens zwei Kinder von $\mathrm{kgV}(U)$ einen M–Wert gleich 1 haben, im Widerspruch zu $\mathrm{kgV}(U) \notin W$.

Sei $v \in W \setminus \{\mathrm{kgV}(U)\}$. Wir bestätigen $\mathrm{depth}(v) > \mathrm{depth}(\mathrm{kgV}(U))$, woraus sofort $\mathrm{kgV}(U) = v_{\mathrm{kgV}}$ folgt. Für $v \in U$ ist dies nach Lemma 3.2.2 erfüllt. Sei also $v \notin U$. Dann ist der Knoten in Zeile (13) zu W hinzugefügt worden, d.h. $\mathrm{outdeg}(v) > 1$ und $M(u_1) = M(u_2) = 1$ für zwei Kinder u_1, u_2 von v. Somit enthalten die Teilbäume T_{u_1}, T_{u_2} jeweils einen Knoten aus U. Da $v \neq \mathrm{kgV}(U)$, muß also $\mathrm{depth}(v) > \mathrm{depth}(\mathrm{kgV}(U))$ gelten.

Die geforderte Zeit– und Prozessorkomplexität folgt aus Lemma 3.2.3 und 3.2.4. ■

3.3 Berechnung einer Blätter–Eliminationsordnung

Sequentiell läßt sich eine Blätter–Eliminationsordnung in Linearzeit berechnen, da jede perfekte Eliminationsordnung (p.e.o.) eine Blätter–Eliminationsordnung ist und eine p.e.o. in Linearzeit berechnet werden kann (vgl. [RTL76] — **LexBFS**, [TY84] — **MCS**).

Es sei die Funktion $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ definiert durch $f(x, y) := \max(x, y + 1)$.

Algorithmus PBEO
Eingabe : Ein Baum $T = (V, E)$.
Ausgabe : Eine Blätter–Eliminationsordnung gegeben durch eine bijektive Abbildung
$\sigma : \{1, \ldots, |V|\} \to V$.

(1) **begin**
(2) Sei w ein beliebiger Knoten aus T;
(3) $T_w := \mathrm{Root{-}Tree}(T, w)$;
(4) $g :=$ **B–ATC**$(T_w, 0, f)$;
(5) Bestimme $\sigma : \{1, \ldots, |V|\} \to V$ so, daß $(g(\sigma(1)), \ldots, g(\sigma(|V|)))$ monoton wachsend ist;
(6) **return**(σ);
(7) **end.**

Satz 3.3.1
Der Algorithmus **PBEO** *berechnet eine Blätter–Eliminationsordnung für einen Baum* $T = (V, E)$ *auf einer* **EREW-PRAM** *in* $O(\log |V|)$ *Zeit unter Benutzung von* $O(|V|)$ *Prozessoren.*

Beweis. Induktiv folgt aus Lemma 2.3.4, daß $g(v)$ für jeden Knoten $v \in V$ die Tiefe des Wurzelbaumes T_v angibt. Somit ist $g(u) < g(v)$ für alle $u \in V(T_v - v)$ und die Knoten aus $V(T_v - v)$ liegen (in σ) links von v. Hieraus folgt, daß σ eine Blätter–Eliminationsordnung für T ist.

Die Zerlegbarkeit des **B–ATC** Problems in Zeile (4) zeigt man genauso, wie wir es bei den früheren **B–ATC** Problemen getan haben. Die Bestimmung von σ in Zeile (5) ist ein Sortierproblem von $|V|$ (nicht notwendig paarweise verschiedenen) Zahlen aus $\{0, \ldots, g(w)\}$. Nach [Col86] geht dies in $O(\log |V|)$ Zeit unter Benutzung von $O(|V|)$ Prozessoren auf einer **EREW-PRAM**. ■

3.4 Dominationsprobleme

Es sei $G = (V, E)$ ein Graph und $r : V \to \mathbb{N}$ eine Knotenbewertung. $D \subseteq V$ heißt eine *r–dominierende Menge* in G, wenn für alle $v \in V$ gilt $d(v, D) \leq r(v)$. Eine r–dominierende Menge D in G heißt *minimal*, wenn keine r–dominierende Menge $D' \subset V$ mit $|D'| < |D|$ existiert. Das **RDS**–*Problem* (*r–dominating set problem*) besteht nun darin, eine minimale r–dominierende Menge in G zu bestimmen.

Oftmals betrachtet man folgende Spezialfälle bzw. Abwandlungen des allgemeinen **RDS**–Problems:

- *Dominating set problem* = **DS**–*Problem.*
  Hier ist $r \equiv 1$.
- *k–dominating set problem* = **KDS**–*Problem.*
  Hier ist r eine konstante Funktion mit $r \equiv k$.
- *Connected r–dominating set problem* = **CRDS**–*Problem.*
  Man sucht eine zusammenhängende r–dominierende Menge mit minimaler Kardinalität.
- *Steiner tree problem* = *Steinerbaumproblem.*
  Gegeben ist eine Teilmenge $D \subseteq V$ von V. Gesucht ist eine minimale zusammenhängende Menge $S \subseteq V$ mit $D \subseteq S$. Das Steinerbaumproblem ist der Spezialfall des **CRDS**–Problems mit

$$r(v) := \begin{cases} 0, & \text{falls } v \in D, \\ |V|, & \text{falls } v \notin D. \end{cases}$$

- *r–dominating path problem* = **RDHP**–*Problem.*
  Eine r–dominierende Menge D, so daß $G(D)$ einen HAMILTONpfad besitzt, nennen wir einen *r–dominierenden Pfad*. Das **RDHP**–Problem besteht nun darin, zu beantworten, ob G einen r–dominierenden Pfad besitzt und falls ja, wie ein r–dominierender Pfad minimaler Kardinalität aussieht.

- *r–dominating cycle problem* = **RDHC**–*Problem.*
  Analog zum **RDHP**–Problem, man braucht nur das Word Pfad durch Kreis ersetzen. *r–dominierenden Kreis*

- *r–dominating clique problem* = **RDC**–*Problem.*
  Eine r–dominierende Menge D nennen wir eine *r–dominierende Clique*, falls $G(D)$ vollständig ist. Das **RDC**–Problem besteht nun darin, zu beantworten, ob G eine r–dominierende Clique besitzt und falls ja, wie eine r–dominierende Clique minimaler Kardinalität aussieht.

Eine Übersicht der Komplexität einiger dieser Probleme für spezielle Graphenklassen gibt Abbildung 3.1 an. Für die Definitionen der vorkommenden Graphenklassen verweisen wir auf das Survey [Bra93] sowie [BDN94]. Wir geben hier nur in Abbildung 3.2 die Inklusionshierarchie der vorkommenden Graphenklassen an.

3.4.1 Das RDC–Problem

In diesem Abschnitt geben wir einen optimalen parallelen Algorithmus an, der entscheidet, ob ein Baum eine r–dominierende Clique besitzt und, falls „ ja “, berechnet.

Da eine Clique in einem Baum nur die Kardinalität 1 und 2 besitzen kann, folgt bereits aus Satz 3.1.4, daß sich das **RDC**–Problem auf einer **CREW–PRAM** in Zeit $O(\log |V|)$ mit $O(|V|^2/\log |V|)$ Prozessoren lösen läßt.

Wir benötigen hierzu die Funktion $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, definiert durch $f(a,b) := \min\{a, b-1\}$.

Algorithmus PRDC
Eingabe : Ein Baum $T = (V, E)$ und eine Abbildung $r : V \to \mathbb{N}$.
Ausgabe : Eine r–dominierende Clique minimaler Kardinalität (falls existent) oder 'Nein'.

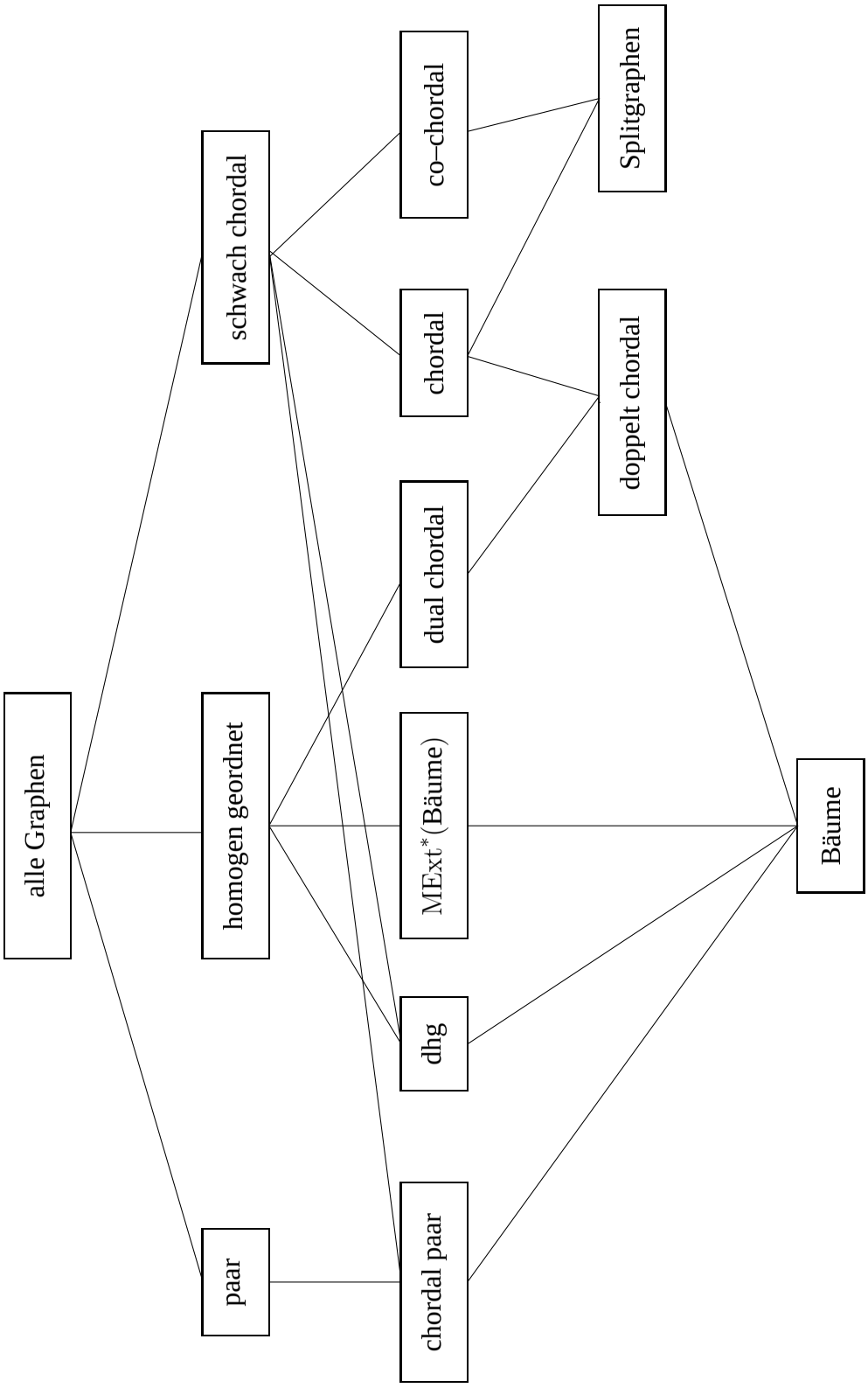
(1) **begin**
(2) Sei w ein beliebiger Knoten aus T;
(3) $T_w := \mathrm{Root{-}Tree}(T, w)$;
(4) $L :=$ **B–ATC**(T_w, r, f);
(5) Bestimme einen Knoten v mit $L(v) = 0$;
(6) **if** (kein solcher Knoten v existiert) **or** $(v = w)$
(7) **then return**$(\{w\})$
(8) **else begin**
(9) $u := \mathrm{parent}(v)$;
(10) $T(u) := \mathrm{Root{-}Tree}(T(V \setminus V(T_v)), u)$;
(11) $M :=$ **B–ATC**$(T(u), r, f)$;

[2] In paaren Graphen enthält jede Clique höchstens zwei Elemente.

Klasse	Erkennung	RDS–Probl.	RDC–Probl.		CRDS–Probl.
			Existenz	Berechnung	
alle Graphen	**LIN** (trivial)	$\mathbb{NP}$–vollst. (s. schwach chordal)	$\mathbb{NP}$–vollst. (s. schwach chordal)		$\mathbb{NP}$–vollst. ([GJ79])
schwach chordal	$O(n^4)$ ([HHM89])	$\mathbb{NP}$–vollst. (s. chordal)	$\mathbb{NP}$–vollst. ([BK87])		$\mathbb{NP}$–vollst. (s. chordal)
homogen geordnet	$O(n^3)$ ([BDN94])	?	$O(n^2)$ ([DN95])		$O(n^2)$ ([DN95])
distanz erblich	**LIN** ([HM90])	?	**LIN** ([Dra94])		**LIN** ([BD94])
dual chordal	**LIN** ([BDCV93])	**LIN** ([BCD94])	**LIN** ([DB94], [BCD94])		$O(E(G^2))$ ([BCD94])
doppelt chordal	**LIN** (s. chordal, dual chordal)	**LIN** (s. dual chordal)	**LIN** (s. dual chordal)		**LIN** ([Dra93])
chordal	**LIN** ([RTL76], [TY84])	$\mathbb{NP}$–vollst. (s. Splitgraphen)	$O(nm)$ ([KDL94], [DB94])	$\mathbb{NP}$–vollst. (s. Splitgraphen)	$\mathbb{NP}$–vollst. ([CP84])
Split-graphen	**LIN** ([Gol80])	$\mathbb{NP}$–vollst. ([BK87])	$O(nm)$ (s. chordal)	$\mathbb{NP}$–vollst. ([BK87])	$\mathbb{NP}$–vollst. ([BK87])
chordal paar	$O(\min(m \log n, n^2))$ ([Spi93], [PT86], [Lub87])	$\mathbb{NP}$–vollst. ([MB87])	$\in \mathbb{P}$ (trivial[2])		$\mathbb{NP}$–vollst. ([MB87])

Abbildung 3.1: Dominationsprobleme auf verschiedenen Graphenklassen

Abbildung 3.2: Inklusionshierarchie einiger Graphenklassen



(12) **if** $M(x) = 0$ für einen Knoten $x \neq u$ aus $T(u)$
(13) **then return**('Nein')
(14) **else if** $M(u) = 0$
(15) **then return**$(\{u, v\})$
(16) **else return**$(\{v\})$;
(17) **end**
(18) **end.**

Für die Korrektheit benutzen wir einen Spezialfall eines Lemmas aus [BCD94].

Lemma 3.4.1
Sei $T = (V, E)$ ein Baum und $r : V \to \mathbb{N}$ eine Knotenbewertung. Weiter sei v ein Blatt in T mit $r(v) > 0$ und u der Nachbar von v. Dann hat T genau dann eine r–dominierende Clique, wenn $T' := T - v$ eine r'–dominierende Clique besitzt, wobei

$$r'(x) := \begin{cases} \min\{r(u), r(v) - 1\}, & \text{falls } x = u, \\ r(x), & \text{falls } x \neq u. \end{cases}$$

Ist D eine r'–dominierende Clique in T', so auch eine r–dominierende Clique in T. ■

Lemma 3.4.2
Es sei $T_w = (V, E)$ ein Wurzelbaum und $r : V \to \mathbb{N}$ eine Knotenbewertung. Dann ist das **B–ATC** *Problem $L :=$* **B–ATC**(T, r, f) *zerlegbar, und es gilt für alle $v \in V$*

$$L(v) = \min_{u \in T_v}(r(u) - d(u, v)). \tag{1}$$

Beweis. Die Zerlegbarkeit des **B–ATC** Problems folgt aus Lemma 3.1.3, da $f = \min(a + 0, b + (-1))$. Die zweite Behauptung folgt induktiv aus Lemma 2.3.4. ■

Satz 3.4.3
Der Algorithmus **PRDC** *löst das* **RDC***–Problem für einen Baum $T = (V, E)$ auf einer* **EREW-PRAM** *in $O(\log |V|)$ Zeit unter Benutzung von $O(|V|/\log |V|)$ Prozessoren.*

Beweis. Ist nach Ausführung von (4) $L(v) > 0$ für alle $v \in V \setminus \{w\}$, so folgt aus Lemma 3.4.2, daß die Wurzel den gesamten Baum r–dominiert.

Also nehmen wir an, daß es mindestens einen Knoten v aus $V \setminus \{w\}$ gibt, dessen L–Wert gleich 0 ist. Aus Lemma 3.4.2 folgt dann, daß $L(u) > 0$ für alle $u \in V(T_v - v)$. Nach Lemma 3.4.1 hat T genau dann eine r–dominierende Clique, wenn $T' := T(V \setminus V(T_v - v))$ eine r'–dominierende Clique besitzt, wobei

$$r'(u) := \begin{cases} 0 & \text{falls } u = v, \\ r(u) & \text{falls } u \neq v. \end{cases}$$

Besitzt T' eine r'–dominierende Clique C, so ist natürlich $v \in C$. Da v ein Blatt in T' ist, brauchen wir nur zu prüfen, ob $\{v\}$ oder $\{v, \text{parent}(v)\}$ den Baum T' r'–dominieren. Dies führen wir in den Zeilen (9) bis (16) durch.

Die geforderte Zeit– und Prozessorkomplexität folgt aus Lemma 3.4.2, da das **B–ATC** Problem in den Zeilen (4) und (11) zerlegbar ist. ■

3.4.2 Das CRDS–Problem

In diesem Abschnitt geben wir einen optimalen parallelen Algorithmus an, der für einen Baum eine minimale zusammenhängende r–dominierende Menge berechnet.

Sei hierzu die Funktion $f : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ wie im letzten Abschnitt definiert, d.h. $f(a, b) := \min\{a, b - 1\}$.

Algorithmus PCRDS
Eingabe : Ein Baum $T = (V, E)$ und eine Abbildung $r : V \to \mathbb{N}$.
Ausgabe : Eine minimale zusammenhängende r–dominierende Menge.

(1) **begin**
(2) Sei w ein beliebiger Knoten aus T;
(3) $T_w := \mathrm{Root{-}Tree}(T, w)$;
{ Schritt 1 }
(4) $L :=$ **B–ATC**(T_w, r, f);
(5) $L_0 := \{v \in V : L(v) = 0\}$;
(6) **if** $L_0 = \emptyset$ **then return**$(\{w\})$; **exit**;
{ Schritt 2 }
(7) $x := \mathrm{kgV}(L_0)$;
(8) $L_1 := L_0 \cup \{v \in V(T_x) : L(v) < 0\}$;
{ Schritt 3 }
(9) **if** $x \neq w$
(10) **then begin**
(11) $T'' := T(V \setminus V(T_x - x))$;
(12) $T''_x := \mathrm{Root{-}Tree}(T'', x)$;
(13) **for all** $v \in V(T'')$ **in parallel do**
(14) **if** $v = x$
(15) **then** $r''(v) := 0$
(16) **else** $r''(v) := r(v)$;
(17) $L'' :=$ **B–ATC**(T''_x, r'', f);
(18) $D := L_1 \cup \{v \in V(T'') : L''(v) \leq 0\}$;
(19) **end**
(20) **else** $D := L_1$;
(21) **return**(D);
(22) **end.**

Für die Korrektheit benutzen wir einen Spezialfall eines Lemmas aus [BCD94].

Lemma 3.4.4
Sei $T = (V, E)$ ein Baum und $r : V \to \mathbb{N}$ eine Knotenbewertung. Weiter sei v ein Blatt in T mit $r(v) > 0$ und u der Nachbar von v. Ist D eine minimale zusammenhängende r'–dominierende Menge in $T' = T - v$, so ist D eine minimale zusammenhängende r–dominierende Menge in T, wobei

$$r'(x) := \begin{cases} \min\{r(u), r(v) - 1\}, & \text{falls } x = u, \\ r(x), & \text{falls } x \neq u. \end{cases}$$

■

Satz 3.4.5
Der Algorithmus **PCRDS** *ist korrekt und läßt sich auf einer* **EREW-PRAM** *optimal parallel implementieren.*

Beweis. Zunächst beweisen wir die Korrektheit des Algorithmus. Der Algorithmus geht in drei Schritten vor.

Der erste Schritt umfaßt die Zeilen (4)–(6). Hier wird zunächst ein **B–ATC** Problem gelöst, das uns schon beim Algorithmus **PRDC** begegnet ist (siehe Lemma 3.4.2). In Zeile (5) werden dann die Knoten mit L–Wert 0 zur Menge L_0 zusammengefaßt. Gilt $L_0 = \emptyset$, so folgt, daß w den Baum T r–dominiert (Lemma 3.4.2). Ist $v \in L_0$, so haben alle Knoten im Teilbaum T_v (außer v natürlich) einen positiven L–Wert, werden also durch v r–dominiert. Sei $A := \bigcup_{v \in L_0} V(T_v - v)$, dann gilt nach Lemma 3.4.4 und Induktion: Eine minimale zusammenhängende r'–dominierende Menge in $T' := T(V \setminus A)$ ist eine minimale zusammenhängende r–dominierende Menge in T, wobei

$$r'(v) := \begin{cases} 0, & \text{falls } v \in L_0, \\ r(v), & \text{falls } v \in T' - L_0. \end{cases}$$

Also brauchen wir im folgenden nur eine minimale zusammenhängende r'–dominierende Menge D in T' berechnen.

Da $r'(v) = 0$ für alle $v \in L_0$, ist $L_0 \subseteq D$, und wegen des Zusammenhangs von D ist sogar der gesamte von $x = \mathrm{kgV}(L_0)$ (in T'_w) aufgespannte Teilbaum in D enthalten, d.h. $L_1 := \bigcup_{u \in L_0} P(u, x) \subseteq D$. L_1 wird in Zeile (8) berechnet, denn $u \in L_1$ genau dann, wenn $u \in V(T_x)$ und $L(u) \leq 0$. Nach Ende des zweiten Schrittes verbleibt also nur noch, eine minimale zusammenhängende r''–dominierende Menge D'' für den Baum $T'' := T(V \setminus V(T_x - x))$ zu bestimmen, wobei

$$r''(v) := \begin{cases} 0, & \text{falls } v = x, \\ r(v), & \text{falls } v \neq x. \end{cases}$$

Denn dann ist $D'' \cup L_1$ eine minimale zusammenhängende r–dominierende Menge in T.

Diese Berechnung führen wir im dritten Schritt durch. Hierzu zeigen wir zunächst:

$(*)$ Es gibt einen Knoten $z \in P(x, w)$ so, daß $D'' = P(x, z)$ ist.

Wir wissen aus dem ersten Schritt, daß alle Knoten aus T'', die nicht auf dem Pfad $P(x, w)$ liegen, nach Ausführung von (4) einen positiven L–Wert besitzen. Daher haben sie auch einen positiven L''–Wert (L'' wird in Zeile (17) berechnet). $(*)$ folgt somit aus Lemma 3.4.4 und Induktion.

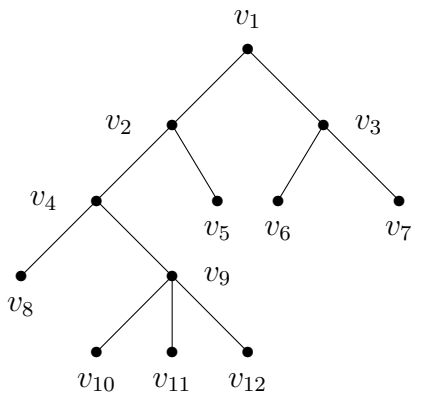
Wie der Knoten z aussieht, steht nach Ausführung von Zeile (17) fest: z ist der eindeutig bestimmt Knoten aus $P(x, w)$ mit $L''(z) = 0$. Damit haben wir mit $D := L_1 \cup P(x, z)$ eine minimale zusammenhängende r–dominierende Menge in T gefunden.

Komplexität: Das **B–ATC** Problem in den Zeilen (4) und (17) ist nach Lemma 3.4.2 zerlegbar und somit optimal parallel lösbar. Der kleinste gemeinsame Vorgänger von L_0 läßt sich nach Korollar 3.2.5 optimal parallel berechnen. ■

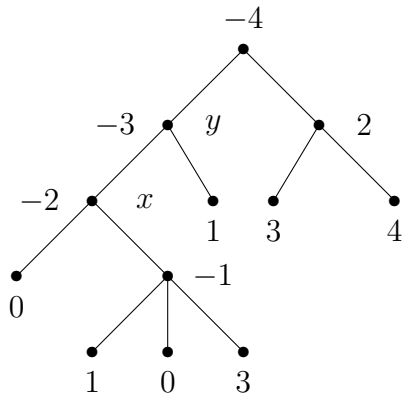
Ein Beispiel für den Algorithmus **PCRDS** ist in Abbildung 3.3 angegeben.

Abbildung 3.3: Beispiel zum Algorithmus **PCRDS**

i	1	2	3	4	5	6	7	8	9	10	11	12
$r(v_i)$	2	3	2	2	1	3	4	0	2	1	0	3



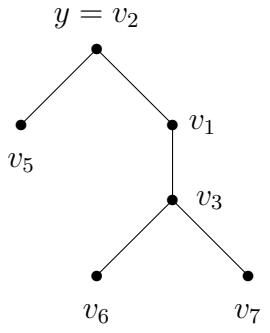
Der Baum T.



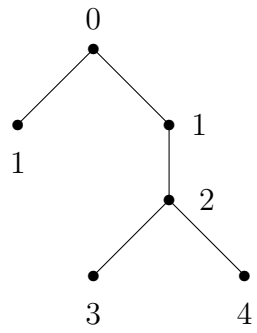
L–Werte für T aus Zeile (4)

$L_0 = \{v_8, v_{11}\}$,

$L_1 = L_0 \cup \{v_4, v_9\}$



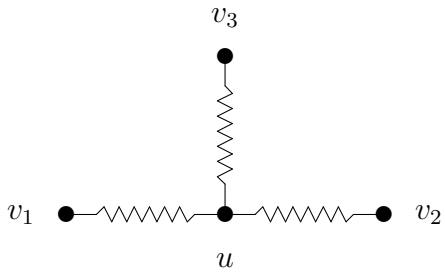
Der Baum $T(y)$.



L–Werte für $T(y)$ nach Zeile (10)

Eine minimale zusammenhängende r–dominierende Menge in T ist $D = \{v_2, v_4, v_8, v_9, v_{11}\}$.

Abbildung 3.4: Skizze zum Beweis von Satz 3.4.6



3.4.3 Das RDHP–Problem

Eine Teilmenge $V' \subseteq V$ eines Graphen $G = (V, E)$ heißt ein *Pfad* in G, wenn es eine Anordnung σ der Knoten aus V' so gibt, daß σ ein Pfad in G ist.

Ist $T = (V, E)$ ein Baum und $V' \subseteq V$, so gilt: $G(V')$ besitzt einen HAMILTONpfad genau dann, wenn V' ein Pfad in T ist.

Satz 3.4.6
Es sei $T = (V, E)$ ein Baum, $r : V \to \mathbb{N}$ eine Knotenbewertung und $D \subseteq V$ eine minimale zusammenhängende r–dominierende Menge in T. Dann gelten folgende Aussagen:

(1) Ist D ein Pfad in T, so ist D ein minimaler r–dominierender Pfad in T.

(2) T besitzt genau dann einen r–dominierenden Pfad, wenn D ein Pfad in T ist.

Beweis. (1) ist trivial, da jeder Pfad einen zusammenhängenden Teilgraphen in T induziert. Wir brauchen also nur noch „$\Longrightarrow$“ von (2) zu bestätigen. Dies tun wird indirekt. Sei D kein Pfad. Dann gibt es mindestens drei Knoten v_1, v_2, $v_3 \in D$, die Blätter in $G(D)$ sind. Weiter sei u der eindeutig bestimmte Knoten aus $P(v_1, v_2) \cap P(v_1, v_3) \cap P(v_2, v_3)$, vgl. auch Abbildung 3.4. Sei T_{v_i} der Teilbaum der Nachfolger von v_i in T_u inklusive v_i, $i = 1, 2, 3$. Da T eine minimale zusammenhängende r–dominierende Menge ist, muß jede zusammenhängende r–dominierende Menge jeweils mindestens einen Knoten aus den Teilbäumen T_i für $i = 1, 2, 3$ enthalten. Somit kann es also insbesondere keinen r–dominierenden Pfad in T geben. ■

Korollar 3.4.7
Das **RDHP***–Problem läßt sich für einen Baum $T = (V, E)$ optimal auf einer* **EREW-PRAM** *lösen.*

Beweis. Nach Satz 3.4.6 brauchen wir nur eine minimale zusammenhängende r–dominierende Menge D berechnen (dies ist nach Satz 3.4.5 optimal möglich) und anschließend überprüfen, ob D ein Pfad in T ist. Diesen Test führen wir wie folgt durch: Wir berechnen zunächst für jeden Knoten $d \in D$ den Grad $\deg_{G(D)}(d)$ von d in $G(D)$. Dies ist ein gewichtetes List Ranking Problem, läßt sich also nach Satz 1.2.1 optimal durchführen. Anschließend berechnen wird die Anzahl l der Knoten aus D, die Grad 1 in $G(D)$ besitzen. Dann gilt: D ist genau dann ein Pfad in G, wenn $l \leq 2$ ist. ■

3.4.4 Das RDS–Problem

Das **RDS**–Problem ist wesentlich schwieriger parallel zu lösen als das **RDC**– oder **CRDS**–Problem. Sequentiell läßt es sich sogar für dual chordale Graphen (**Bäume** $\subset$ **dual chordal**) in Linearzeit lösen, wie in [BCD94] gezeigt wurde. Will man den Algorithmus, um ihn zu parallelisieren, auf ein **B–ATC** Problem zurückführen, so ist die dabei vorkommende Funktion f so kompliziert, daß dieses **B–ATC** Problem nicht zerlegbar zu sein scheint ([HY90], dort wurde das **KDS**-Problem für Bäume behandelt).

Die Funktion $f : [-R, R] \times [-R, R] \to [-R, R]$ ist gegeben durch[3]

$$f(a,b) := \begin{cases} a, & \text{falls } a \geq 0 \text{ und } (-R \leq b \leq -a-1 \text{ oder } a \leq b \leq R), \\ b+1, & \text{falls } a \geq 0 \text{ und } (-a \leq b \leq -1 \text{ oder } 0 \leq b \leq a-1), \\ a, & \text{falls } a < 0 \text{ und } (-R \leq b \leq a-1 \text{ oder } -a \leq b \leq R), \\ b+1, & \text{falls } a < 0 \text{ und } (a \leq b \leq -1 \text{ oder } 0 \leq b \leq -a-1). \end{cases}$$

Algorithmus PRDS
Eingabe : Ein Baum $T = (V, E)$ und eine Abbildung $r : V \to \mathbb{N}$.
Ausgabe : Eine minimale r–dominierende Menge.

(1) **begin**
(2) Sei w ein beliebiger Knoten aus T;
(3) $T_w := \text{Root}-\text{Tree}(T, w)$;
(4) $R := \max\{r(v) : v \in V\}$;
(5) $L :=$ **B–ATC**(B_w, r, f);
(6) $D := \{v \in V : L(v) = 0\}$;
(7) **if** $L(w) < 0$
(8) **then** $D := D \cup \{w\}$;
(9) **return**(D);
(10) **end.**

Satz 3.4.8
Der Algorithmus **PRDS** *ist korrekt und läßt sich auf einer* **CREW–PRAM** *in Zeit* $O(\log n \log\log n)$ *unter Benutzung von* $O(n)$ *Prozessoren implementieren.*

Beweis. Korrektheit: Man braucht nur beim Beweis zum **KDS**-Problem aus [HY90] den k–Wert durch die r–Werte der Knoten ersetzen.

Komplexität: Dies wurde bereits in [HY90] gezeigt. ■

3.5 Übersicht der Algorithmen dieses Kapitels

Die folgende Tabelle gibt eine Übersicht der Komplexität der Algorithmen dieses Kapitels. Dabei bedeutet OPT Optimalität, d.h. sequentiell Linearzeit und parallel $O(\log n)$ Zeit mit

[3] Mit $[-R, R]$ ist hier natürlich $\{-R, -R+1, \ldots, R-1, R\}$ gemeint.

$O(n/\log n)$ Prozessoren, wenn n die Länge der Eingabe bezeichnet.

Problem	**sequentielle Zeit**	**parallel**		
		Zeit	**Prozessoren**	**PRAM–Typ**
SSSP	OPT			**EREW**
APSP	$O(n^2)$	$O(\log n)$	$O(n^2/\log n)$	**EREW**
DistSet	OPT			**EREW**
kgV	OPT			**EREW**
BEO	OPT	$O(\log n)$	$O(n)$	**EREW**
RDC	OPT			**EREW**
CRDS	OPT			**EREW**
RDHP	OPT			**EREW**
RDS	OPT	$O(\log n \log\log n)$	$O(n)$	**CREW**

Kapitel 4

Modulare Erweiterungen und Zerlegungen

Zunächst definieren wir in den ersten beiden Abschnitten grundlegende Begriffe, die zur Untersuchung von modularen Erweiterungen einer Graphenklasse benötigt werden.

Anschließend führen wir die Graphenklasse **M** ein, die technisch für die Beweise der folgenden Abschnitte hilfreich ist.

In Abschnitt 4.4 zeigen wir dann, daß modulare Erweiterungen einer Graphenklasse **G**, die bezüglich Primgraphbildung abgeschlossen ist, genauso effizient sequentiell und parallel erkannt werden können wie Graphen aus **G**.

Im Abschnitt 4.5 untersuchen wir das **RDS**–Problem auf modularen Erweiterungen von Bäumen. Hierbei zeigen wir, daß es sich sequentiell in Linearzeit lösen läßt (Satz 4.5.5). Es sei angemerkt, daß auch das **CRDS**– und **RDC**–Problem für modulare Erweiterungen von Bäumen effizient sequentiell und parallel lösbar sind ([NS95]).

Wir schließen dieses Kapitel mit einigen Problemen, die auf modularen Erweiterungen $\mathbb{NP}$–vollständig sind.

4.1 Definitionen

Es seien $G = (V, E)$ ein Graph und V_1, V_2 nichtleere disjunkte Teilmengen von V. Wir schreiben $V_1 \bowtie V_2$ (bzw. $V_1 \parallel V_2$), falls (in G) jeder Knoten aus V_1 mit jedem Knoten aus V_2 adjazent ist (bzw. falls $V_1 \bowtie V_2$ in $\overline{G}$ gilt). G heißt *join–zerlegbar* (bzw. *anti join–zerlegbar*), wenn sich V so in zwei Mengen V_1, V_2 partitionieren läßt, daß $V_1 \bowtie V_2$ (bzw. $V_1 \parallel V_2$) gilt.

Eine nichtleere Menge $M \subseteq V$ heißt *Modul* (oder *modulare Menge*, *homogene Menge*) (in G), wenn alle Knoten aus M in $V \setminus M$ die gleiche Nachbarschaft besitzen, d.h.

$$N(m_1) \cap (V \setminus M) = N(m_2) \cap (V \setminus M) \quad \text{für alle } m_1, m_2 \in M.$$

Es sei M ein Modul. M heißt *echt*, wenn $M \neq V$. M heißt *trivial*, falls $M = V$ oder $M = \{v\}$, $v \in V$. Graphen, die nur triviale Module besitzen, nennen wir *Primgraphen*. Die dazugehörige Graphenklasse bezeichnen wir mit **Prim**.

Es sei M ein (in G) nichttrivialer Modul und $v_M \in M$. Dann sei $\mathrm{MRed}(G, M, v_M)$ derjenige Graph, den man erhält, wenn man den Modul M in G zum repräsentierenden Knoten v_M

reduziert, d.h. $\mathrm{MRed}(G, M, v_M) = G - (M \setminus \{v_M\}) = G(V \setminus (M \setminus \{v_M\}))$.

Umgekehrt kann man in einem Graphen wie folgt einen Knoten modular erweitern: Seien $v \in V$ ein Knoten und H ein Graph mit $V(H) \cap V = \emptyset$. Dann sei $G' := (V', E') := \mathrm{MExt}(G, v, H)$ derjenige Graph, den man erhält, wenn man den Graphen H in G „nachbarschaftstreu“ in den Knoten v einsetzt, d.h.

$$V' := (V \setminus \{v\}) \cup V(H)$$

und

$$E' := \{e \in E : v \notin e\} \cup E(H) \cup \{hx : h \in V(H), x \in V, xv \in E\}.$$

Es sei **G** eine Graphenklasse. Dann sei $\mathrm{MExt}^*(\mathbf{G})$ die wie folgt rekursiv definierte Graphenklasse:

(1) Jeder Graph $G \in \mathbf{G}$ gehört zu $\mathrm{MExt}^*(\mathbf{G})$.

(2) Ist $G = (V, E) \in \mathrm{MExt}^*(\mathbf{G}) \setminus \{K_1\}$, $v \in V$, H ein Graph mit $V(H) \cap V = \emptyset$, so gehört auch der Graph $\mathrm{MExt}(G, v, H)$ zu $\mathrm{MExt}^*(\mathbf{G})$.[1]

(3) Weitere Graphen enthält $\mathrm{MExt}^*(\mathbf{G})$ nicht.

$\mathrm{MExt}^*(\mathbf{G})$ kann man als transitive Hülle der Graphenklasse **G** bzgl. modularer Erweiterungen ansehen.

4.2 Der einem Graphen zugehörige modulare Baum und Primgraph

Zwei Module M_1, M_2 in G *überlappen* sich, wenn $M_1 \cap M_2$, $M_1 \setminus M_2$ und $M_2 \setminus M_1$ nichtleere Mengen sind. Nicht überlappende Module nennen wir *überlappungsfrei*. Ein Modul M heißt *überlappungsfrei*, wenn für jeden Modul M' die Module M, M' überlappungsfrei sind. In jedem Graphen sind $\{v\}$, $v \in V$, und V triviale Beispiele für überlappungsfreie Module.

Ist $M \neq V$ ein überlappungsfreier Modul, so existiert genau ein überlappungsfreier Modul $M' \supset M$ mit minimaler Kardinalität.[2] Setzen wir $\mathrm{parent}(M) := M'$, so erhalten wir wie folgt einen Wurzelbaum T_G mit Wurzel V: Die Knotenmenge von T_G bilden die überlappungsfreien Module und die Kanten sind durch die parent–Funktion gegeben. T_G nennen wir den *modularen Baum* zu G. (T_G ist ein Baum, da jedes $M \in V(T_G)$ nach Konstruktion der parent–Funktion durch einen Pfad (in T_G) mit V verbunden ist. Die Kreisfreiheit ist durch $\mathrm{parent}(M) \supset M$, $M \in V(T_G) \setminus \{V\}$, gesichert.) Der modulare Baum kann effizient berechnet werden, wie der folgende Satz bestätigt.

Satz 4.2.1
Es sei G ein Graph. Dann kann der modulare Baum T_G zu G in der folgenden Komplexität berechnet werden:

[1] $G \neq K_1$ ist hierbei wesentlich, da $\mathrm{MExt}((\{x\}, \emptyset), x, H) = H$.

[2] Denn wären M_1', M_2' zwei verschiedene (bzgl. der Kardinalität) minimale überlappungsfreie Moduln mit $M_i' \supset M$, $i = 1, 2$, so würden sich wegen $M_1' \not\subseteq M_2'$, $M_2' \not\subseteq M_1'$ und $M \subseteq M_2' \cap M_1'$ die Module M_1' und M_2' überlappen.

- *([MS94])*
  Sequentiell in Linearzeit $O(|V| + |E|)$.

- *([Dah95])*
  Parallel auf einer **CRCW–PRAM** *in Zeit* $O(\log^2 |V|)$ *mit* $O(|V| + |E|)$ *Prozessoren.* ■

Es sei $\mathcal{M}(G)$ der Hypergraph der echten maximalen Module in G. Zur Definition des Primgraphen eines Graphen benötigen wir folgendes

Lemma 4.2.2 ([DMS88])
Es seien $G = (V, E)$ *ein Graph und* $M_1, M_2 \in \mathcal{M}(G)$ *mit* $M_1 \neq M_2$. *Dann gilt:*

(1) Aus $M_1 \cap M_2 \neq \emptyset$ *folgt:*

- $M_1 \cup M_2 = V$,
- $M_1 \cap M_2$ *und* $V \setminus (M_1 \cap M_2)$ *sind Module in* G, *d.h. insbesondere, daß* G *join–zerlegbar oder anti join–zerlegbar ist.*

(2) Ist $\overline{G}$ *nicht zusammenhängend, so gilt*

$$\mathcal{M}(G) = \{V \setminus V_i : i = 1, \ldots, k\},$$

wobei $V_1, \ldots, V_k$ *die Zusammenhangskomponenten von* $\overline{G}$ *bezeichnen.* ■

Es sei $G = (V, E)$ ein Graph. Dann sei der Primgraph $\mathrm{Prim}(G)$ wie folgt definiert:

(1) Ist $G = K_1$, so sei $\mathrm{Prim}(G) := K_1$.

(2) Ist $\mathcal{M}(G)$ eine Partition von V, so sei $\mathrm{Prim}(G)$ derjenige Graph, den man erhält, wenn man jeden echten maximalen Modul in G zu einem repräsentierenden Knoten reduziert. Dann ist wegen der Maximalität der (echten) Module aus $\mathcal{M}(G)$ der Graph $\mathrm{Prim}(G)$ natürlich ein Primgraph.

(3) Ist $\mathcal{M}(G)$ keine Partition von G und $G \neq K_1$, so ist G nach Lemma 4.2.2 (1) join–zerlegbar oder anti join–zerlegbar. Wir setzen dann

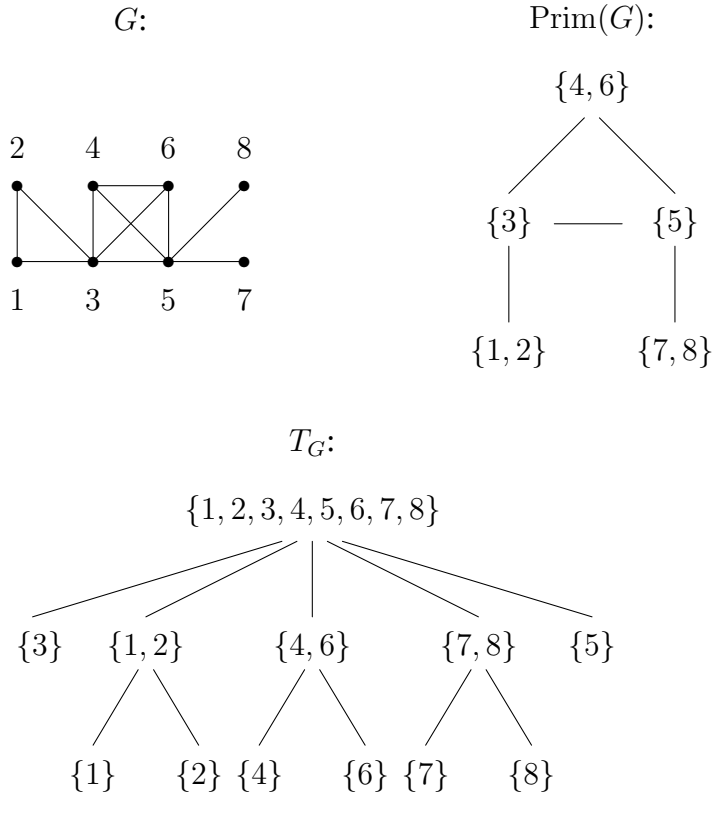
$$\mathrm{Prim}(G) := \begin{cases} K_2, & \text{falls } G \text{ join–zerlegbar,} \\ 2K_1, & \text{falls } G \text{ anti join–zerlegbar.} \end{cases}$$

Ein Beispiel für den zugehörigen modularen Baum und Primgraphen eines Graphen ist in Abbildung 4.1 dargestellt.

4.3 Die Graphenklasse M

In diesem Abschnitt untersuchen wir, wann $\mathcal{M}(G)$ ein dualer Hyperbaum ist. Die Graphenklasse aller Graphen G, für die $\mathcal{M}(G)$ ein dualer Hyperbaum ist, bezeichnen wir mit **M**.

Abbildung 4.1: Beispiel für den zugehörigen modularen Baum und Primgraphen eines Graphen



Lemma 4.3.1
Es sei $G = (V, E)$ ein zusammenhängender Graph. Dann ist $\mathcal{M}(G)$ ein dualer Hyperbaum oder G ist join–zerlegbar.

Beweis. Wir unterscheiden zwei Fälle:

Fall 1. Alle Elemente aus $\mathcal{M}(G)$ sind paarweise disjunkt.
Dann ist $\mathcal{M}(G)$ trivialerweise ein dualer Hyperbaum, da jeder Baum mit der Knotenmenge $\mathcal{M}(G)$ einen unterliegenden Baum für $\mathcal{M}(G)$ darstellt.

Fall 2. Es existieren echte Module M_1, M_2 mit $M_1 \cap M_2 \neq \emptyset$.
Dann sind nach Lemma 4.2.2 (1) die Mengen $V_1 := M_1 \cap M_2$ und $V_2 := V \setminus (M_1 \cap M_2)$ Module in G. Da G zusammenhängend ist, folgt $V_1 \bowtie V_2$. ■

Korollar 4.3.2
Es sei G ein zusammenhängender Graph. Ist $c(\overline{G}) \leq 2$, so ist $\mathcal{M}(G)$ ein dualer Hyperbaum.

Beweis. Ist $\overline{G}$ zusammenhängend, so ist G nicht join–zerlegbar und somit $\mathcal{M}(G)$ nach Lemma 4.3.1 ein dualer Hyperbaum. Ist $c(\overline{G}) = 2$, so ist $\mathcal{M}(G)$ nach Lemma 4.2.2 (2) zweielementig und somit trivialerweise ein dualer Hyperbaum. ■

Es sei $\mathcal{H} = (V, \mathcal{E})$ ein Hypergraph. Der *„2–section–graph"* $2\mathrm{SEC}(\mathcal{H})$ von $\mathcal{H}$ ist dann der folgende Graph: $V(2\mathrm{SEC}(\mathcal{H})) := V$ und für u, $v \in V$ gilt $uv \in E(2\mathrm{SEC}(\mathcal{H}))$ genau dann, wenn $u \neq v$ und $\{u, v\} \subseteq e$ für ein $e \in \mathcal{E}$ gilt.

$\mathcal{H}$ heißt *konform*, wenn jede vollständige Menge (Clique) in $2\mathrm{SEC}(\mathcal{H})$ in einer Hyperkante von $\mathcal{H}$ enthalten ist.

Ein Graph G heißt *chordal*, wenn G keinen C_n, $n \geq 4$, als induzierten Teilgraphen enthält. Dann gilt folgende wichtige Charakterisierung für duale Hyperbäume:

Satz 4.3.3 ([Duc76], [Fla78])
Es sei $\mathcal{H}$ ein Hypergraph. Dann ist $\mathcal{H}$ genau dann ein dualer Hyperbaum, wenn $\mathcal{H}$ konform und $2\mathrm{SEC}(\mathcal{H})$ chordal ist. ■

Lemma 4.3.4
Es sei $G = (V, E)$ ein Graph mit $c(\overline{G}) \geq 3$. Dann ist $2\mathrm{SEC}(\mathcal{M}(G))$ vollständig und $\mathcal{M}(G)$ kein dualer Hyperbaum.

Beweis. Es seien $V_1, \ldots, V_k$ die Zusammenhangskomponenten von $\overline{G}$. Nach Lemma 4.2.2 (2) sind dann $M_i := V \setminus V_i$, $i = 1, \ldots, k$, die maximalen echten Module in G. Seien u, v zwei verschiedene Knoten aus V. Dann gibt es wegen $k \geq 3$ ein $j \in \{1, \ldots, k\}$ mit u, $v \notin V_j$. Also gilt $\{u, v\} \subseteq M_j$ und somit $uv \in E(2\mathrm{SEC}(\mathcal{M}(G)))$. Hieraus folgt die Vollständigkeit von $2\mathrm{SEC}(\mathcal{M}(G))$.

Da V eine Clique in $2\mathrm{SEC}(\mathcal{M}(G))$ ist, die in keiner Hyperkante von $\mathcal{M}(G)$ enthalten ist, erhalten wir aus Satz 4.3.3, daß $\mathcal{M}(G)$ kein dualer Hyperbaum ist. ■

Korollar 4.3.5
Es sei G ein zusammenhängender Graph. Dann ist $\mathcal{M}(G)$ genau dann ein dualer Hyperbaum, wenn $c(\overline{G}) \leq 2$. ■

Für nicht zusammenhängende Graphen, erhält man eine Charakterisierung sofort aus

Lemma 4.3.6
Für einen Graphen G gelten folgende Aussagen:

(1) Ist G nicht zusammenhängend, so ist $\overline{G}$ zusammenhängend.

(2) Es sei $M \subseteq V$ eine Teilmenge von V. Dann ist M genau dann ein Modul in $\overline{G}$, wenn M ein Modul in G ist. ■

Korollar 4.3.7
Es gilt **M** $=$ *co–***M**, *d.h. $\mathcal{M}(G)$ ist genau dann ein dualer Hyperbaum, wenn $\mathcal{M}(\overline{G})$ ein dualer Hyperbaum ist.* ■

Korollar 4.3.8 (Charakterisierungen für die Graphenklasse M)
Sei G ein Graph. Dann sind die folgenden Aussagen paarweise äquivalent:

*(1) $G \in$ **M**.*

(2) G ist entweder zusammenhängend und $c(\overline{G}) \leq 2$ oder $c(G) = 2$.

(3) $\mathcal{M}(G)$ ist eine Partition von V oder $G = K_1$ (d.h. $\mathcal{M}(G) = \emptyset$).

Beweis. Die Beziehungen „$(1) \iff (2)$“ und „$(3) \implies (1)$“ folgen sofort aus den obigen Ergebnissen. Wir zeigen nun „$(1) \implies (3)$“. Sei dazu $G \in$ **M**, $G \neq K_1$.

Fall 1. G ist zusammenhängend.
Dann ist $c(\overline{G}) \leq 2$. Wir nehmen an, $\mathcal{M}(G)$ sei keine Partition von V. Dann impliziert Lemma 4.3.1, daß G join–zerlegbar ist. Somit hat $\overline{G}$ genau zwei Zusammenhangskomponenten, im Widerspruch zu Lemma 4.2.2 (2).

Fall 2. G ist nicht zusammenhängend.
Dann ist $c(G) = 2$ und somit $\mathcal{M}(G) = \mathrm{ZHK}(G)$ eine Partition von V. ■

Bemerkung 4.3.9
Es gelten die folgenden Aussagen:

(1) Bäume und Primgraphen sind Graphen aus **M**.

*(2) co–***M** $\subset \mathrm{MExt}^*(\{2K_1, K_2\}) \subset \mathrm{MExt}^*($**Wälder**$)$.

Beweis. Ein Baum G ist zusammenhängend und wegen der C_3–Freiheit kann $\overline{G}$ nicht mehr als zwei Zusammenhangskomponenten enthalten. (2) folgt sofort aus Lemma 4.2.2 (1) und $K_2 \in \mathrm{MExt}^*(\{2K_1, K_2\}) \cap$ **M**, $P_4 \in \mathrm{MExt}^*($**Wälder**$) \setminus \mathrm{MExt}^*(\{2K_1, K_2\})$. ■

4.4 Erkennung modularer Erweiterungen von Graphen

Es sei **G** eine Graphenklasse. **G** heißt *bezüglich Primgraphbildung abgeschlossen*, wenn für alle $G \in$ **G** gilt $\mathrm{Prim}(G) \in$ **G**. Wir zeigen in diesem Abschnitt, daß man modulare Erweiterungen einer bezüglich Primgraphbildung abgeschlossenen Graphenklasse **G** genauso effizient erkennen kann wie Graphen aus **G**. Der Grund hierfür liegt darin, daß sich der zugehörige Primgraph eines Graphen G aus dem modularen Baum T_G (genauer: aus den Kindern der Wurzel in T_G) effizient berechnen läßt.

Es sei G ein Graph. Dann bezeichne $\mathrm{Mod}(G)$ die Menge aller Module in G.

Lemma 4.4.1
Es sei G ein zusammenhängender Graph. Dann gilt

$$\mathrm{Mod}(G) = \{V\} \cup \bigcup_{M \in \mathcal{M}(G)} \mathrm{Mod}(G(M)).$$

Ist $G \notin \mathbf{M}$ und sind $V_1, \ldots, V_k$ die Zusammenhangskomponenten von $\overline{G}$, so gilt speziell

$$\mathrm{Mod}(G) = \bigcup_{i=1}^{k} \mathrm{Mod}(G(V_i)) \cup \left\{ \bigcup_{i \in I} V_i : I \in \mathcal{P}(\{1, \ldots, k\} \setminus \{\emptyset\}) \right\}.$$

Beweis. Sei M ein echter Modul in G. Dann ist M in einem maximalen echten Modul $M_{\max} \in \mathcal{M}(G)$ enthalten und damit insbesondere ein Modul in $G(M_{\max})$.

Ist umgekehrt M ein Modul in $G(M_{\max})$, so ist M auch ein Modul in $G(M \cup (V \setminus M_{\max}))$, da $M_{\max}$ ein Modul in G ist. Hieraus folgt sofort, daß M ein Modul in G ist.

Sei nun $G \notin \mathbf{M}$ und $V_1, \ldots, V_k$ die Zusammenhangskomponenten von $\overline{G}$. Dann ist $k \geq 3$ nach Korollar 4.3.8. Ist M ein Modul in $G(V_i)$, $i \in \{1, \ldots, k\}$, so ist wegen $M \,||_{\overline{G}}\, V_j$, $j \in \{1, \ldots, k\} \setminus \{i\}$, M auch ein Modul in G. Ebenso leicht sieht man, daß für $I \in \mathcal{P}(\{1, \ldots, k\})$, $I \neq \emptyset$, die Menge $\bigcup_{i \in I} V_i$ ein Modul in G ist.

Seien $i_0, i_1 \in \{1, \ldots, k\}$ mit $i_0 \neq i_1$. Wir nehmen nun an, daß es einen Modul M in G gibt, der

$$M \cap V_{i_0} \neq \emptyset, \qquad M \cap V_{i_1} \neq \emptyset, \qquad M \neq V_{i_0}$$

erfüllt. Da $\overline{G}(V_{i_0})$ zusammenhängend ist, gibt es einen Knoten $x \in V_{i_0} \setminus M$ und einen Knoten $m \in M$ mit $xm \in E(\overline{G})$. Da $x \,||_{\overline{G}}\, M \setminus V_{i_0}$ liefert dies einen Widerspruch zur Modularität von M. ■

Welche Bedeutung die Mengen aus $N_{T_G}(V)$ im Graphen G besitzen, zeigt daß folgende

Lemma 4.4.2
Es sei G ein zusammenhängender Graph. Dann gilt:

(1) Ist $G \in \mathbf{M}$, so ist $N_{T_G}(V) = \mathcal{M}(G)$.

(2) Ist $G \notin \mathbf{M}$, so sind $N_{T_G}(V)$ die Zusammenhangskomponenten von $\overline{G}$.

Beweis. Ist $G \in \mathbf{M}$, so ist nach Korollar 4.3.8 $\mathcal{M}(G)$ eine Partition von G. Aus Lemma 4.4.1 folgt dann sofort, daß $N_{T_G}(V) = \mathcal{M}(G)$ gilt.

Seien $G \notin \mathbf{M}$ und $V_1, \ldots, V_k$ die Zusammenhangskomponenten von $\overline{G}$. Dann gilt $k \geq 3$ nach Korollar 4.3.8. Es sei $i \in \{1, \ldots, k\}$. Ist dann M ein Modul in G mit $M \cap V_i \neq \emptyset$, so folgt nach Lemma 4.4.1 $M \subseteq V_i$ oder $V_i \subseteq M$. Damit ist V_i ein echter überlappungsfreier Modul.

Sei $I \subset \{1, \ldots, k\}$ mit $|I| \geq 2$. Dann existieren $i_0, i_1 \in I$ mit $i_0 \neq i_1$ und ein $i_2 \in \{1, \ldots, k\} \setminus I$. Dann sind $\bigcup_{i \in I} V_i$ und $V_{i_0} \cup V_{i_2}$ überlappende Module. Somit ist nach Lemma 4.4.1

$$N_{T_G}(V) = \{V_1, \ldots, V_k\}$$

bewiesen. ■

Satz 4.4.3
Es sei G ein Graph. Dann können folgende Berechnungen bzw. Entscheidungen sequentiell und parallel in der gleichen Komplexität durchgeführt werden, wie die Berechnung des modularen Baumes zu G aus Satz 4.2.1:

(1) Berechnung von $\mathrm{Prim}(G)$.

(2) Entscheidung, ob $G \in \mathbf{M}$.

(3) Berechnung von $\mathcal{M}(G)$, falls $G \in \mathbf{M}$. Falls $G \notin \mathbf{M}$, so kann $\overline{\mathcal{M}(G)} := \{V \setminus M : M \in \mathcal{M}(G)\}$ berechnet werden.[3]

(4) Berechnung der Zusammenhangskomponenten von $\overline{G}$.

Beweis. Zunächst berechnen wir die Zusammenhangskomponenten $V_1, \ldots, V_k$ von G. Dies geht sequentiell mit Breiten– oder Tiefensuche in Linearzeit ([Bra94]) und parallel auf einer **CRCW–PRAM** in Zeit $O(\log |V|)$ mit $O(|V| + |E|)$ Prozessoren ([SV82]). Wir unterscheiden nun drei Fälle:

Fall 1. $k \geq 3$.
Dann ist G anti join–zerlegbar, denn es gilt:

$\mathrm{Prim}(G) := 2K_1$	(nach Lemma 4.3.6 (2) und Lemma 4.2.2 (1), (2)),
$\mathcal{M}(G) := \{V \setminus V_1, \ldots, V \setminus V_k\}$	(nach Lemma 4.3.6 (2) und 4.2.2 (2)),
$\mathrm{ZHK}(\overline{G}) = \{V\}$	(nach Lemma 4.3.6 (1)),
$G \notin \mathbf{M}$	(nach Korollar 4.3.8).

Fall 2. $k = 2$.
Dann ist G anti join–zerlegbar, und es gilt:

$\mathrm{Prim}(G) := 2K_1$	(nach Lemma 4.3.6 (2) und 4.2.2 (2)),
$\mathcal{M}(G) := \{V_1, V_2\}$	(nach Lemma 4.3.6 (2) und 4.2.2 (2)),
$\mathrm{ZHK}(\overline{G}) = \{V\}$	(nach Lemma 4.3.6 (1)),
$G \in \mathbf{M}$	(nach Korollar 4.3.8).

Fall 3. $k = 1$, d.h. G ist zusammenhängend.
Dann berechnen wir den modularen Baum T_G zu G (Komplexität siehe Satz 4.2.1). Seien $V_1, \ldots, V_k$ die Kinder von V in T_G.

Fall 3.1. $k = 0$.
Dann ist V einelementig (d.h. G ist ein K_1) und somit gilt trivialerweise

$\mathrm{Prim}(G) = K_1$,

$\mathrm{ZHK}(\overline{G}) = \{V\}$,

$\mathcal{M}(G) := \emptyset$,

$G \in \mathbf{M}$.

[3] Dies liegt daran, daß falls $G \notin \mathbf{M}$, die Größe der Menge $\mathcal{M}(G)$ nicht mehr in $O(|V| + |E|)$ liegen muß. So besteht $\mathcal{M}(nK_1)$ z.B. aus n Mengen der Kardinalität $n-1$, d.h. $\mathcal{M}(nK_1)$ hat (bezogen auf n) quadratische Größe.

Fall 3.2. $k = 1$.

Dieser Fall kann nicht auftreten, da die einelementigen Teilmengen von $V \setminus V_1 \neq \emptyset$ nicht in dem von V_1 (in T_G) induziertem Teilbaum liegen können.

Fall 3.3. $k = 2$.

Dann ist G join–zerlegbar ($V = V_1 \bowtie V_2$). Da $k = 2$ ist nach Lemma 4.4.2 $c(\overline{G}) = 2$, somit gilt:

$\mathrm{Prim}(G) := K_2$ (nach Lemma 4.2.2 (2)),

$\mathfrak{M}(G) := \{V_1, V_2\}$ (nach Lemma 4.2.2 (2)),

$\mathrm{ZHK}(\overline{G}) = \{V_1, V_2\}$ (nach Lemma 4.2.2 (2)),

$G \in \mathbf{M}$ (nach Korollar 4.3.8).

Fall 3.4. $k \geq 3$.

Berechne $M_2 := V_2 \cup \ldots \cup V_k$. Dann prüfen wir, ob M_2 ein Modul in G ist. Dies läßt sich sequentiell und parallel optimal mit gewichtetem list–ranking lösen, indem man für alle Knoten $v \in V \setminus M_2$ die Anzahl $a_v := |N(v) \cap M_2|$ berechnet (vgl. Satz 1.2.1). Denn dann ist M_2 genau dann ein Modul, wenn für alle Knoten $v \in V \setminus M_2$ entweder $a_v = 0$ oder $a_v = |M_2|$ gilt.

Fall 3.4.1. M_2 ist ein Modul in G.

Dann gilt $V = M_2 \bowtie V_1$ und aus Lemma 4.4.2 folgt:

$G \notin \mathbf{M}$,

$\mathrm{Prim}(G) := K_2$,

$\mathrm{ZHK}(\overline{G}) = \{V_1, \ldots, V_k\}$,

$\mathfrak{M}(G) := \{V \setminus V_1, \ldots, V \setminus V_k\}$ (nach Lemma 4.2.2 (2)).

Fall 3.4.2. M_2 ist kein Modul in G.

Dann ist $G \in \mathbf{M}$ und $\mathfrak{M}(G) := \{V_1, \ldots, V_k\}$ nach Lemma 4.4.2. $\mathrm{Prim}(G)$ erhalten wir, indem wir jede der Mengen V_i in G durch einen repräsentierenden Knoten aus V_i ersetzen. Dies geht effizient: Man markiert zunächst diejenigen Knoten, die entfernt werden sollen. Anschließend berechnet man für die Knotenliste und den Adjazenzlisten die Teillisten der nicht markierten Knoten (parallel benutzt man hierzu die Verdopplungstechnik (siehe [GR88])). $\overline{G}$ ist zusammenhängend, d.h. $\mathrm{ZHK}(\overline{G}) = \{V\}$, da $G \in \mathbf{M}$ und $|\mathfrak{M}(G)| \geq 3$. ■

Wie sich der Primgraph bei Komplementbildung bzw. bei modularer Reduktion verhält, beantwortet das folgende

Lemma 4.4.4

Es sei G ein Graph. Dann gelten die folgenden Aussagen:

(1) $\mathrm{Prim}(\overline{G}) = \overline{\mathrm{Prim}(G)}$.

(2) Seien $M \in \mathrm{Mod}(G)$ ein echter Modul in G und $v_M \in M$. Wir setzen $G' := \mathrm{MRed}(G, M, v_M)$. Dann gilt:

(a) $\mathrm{Prim}(G) = \mathrm{Prim}(G')$.

(b) $G \in \mathbf{M} \Longrightarrow G' \in \mathbf{M}$. *Die umgekehrte Implikation ist jedoch nicht allgemeingültig.*

Beweis. Die Aussage (1) folgt unmittelbar aus der Definition von $\mathrm{Prim}(G)$ und Lemma 4.3.6 (2).

Nun zum Beweis von (2). Ist $G \notin \mathbf{M}$, so unterscheiden wir gemäß Korollar 4.3.8 zwei Fälle.

Fall 1. $c(\overline{G}) > 2$.
Nach Lemma 4.4.1 folgt dann $c(\overline{G'}) \geq 2$ und somit

$$\mathrm{Prim}(G) = \mathrm{Prim}(G') = K_2.$$

Fall 2. $c(G) > 2$.
Aus (1) und Fall 1 schließen wir

$$\mathrm{Prim}(G) = \overline{\mathrm{Prim}(\overline{G})} = \overline{K_2} = 2K_1.$$

Sei nun $G \in \mathbf{M}$. Da M ein echter Modul in G ist, gibt es ein $M_{\max} \in \mathcal{M}(G)$ mit $M_{\max} \supseteq M$. Im folgenden zeigen wir $\mathcal{M}(G') = \mathcal{M}'$, wobei

$$\mathcal{M}' := (\mathcal{M}(G) \setminus \{M_{\max}\}) \cup \{(M_{\max} \setminus M) \cup \{v_M\}\}.$$

Hieraus folgt dann unmittelbar $G' \in \mathbf{M}$ und $\mathrm{Prim}(G) = \mathrm{Prim}(G')$.

Sei $M' \in \mathcal{M}'$. Dann ist M' ein echter Modul in G'. Wir zeigen, daß M' auch ein maximaler echter Modul ist.

Fall 1. $M' = (M_{\max} \setminus M) \cup \{v_M\}$.
Wäre M' nicht maximal, d.h. $M' \subset M'_{\max}$, $M'_{\max} \in \mathcal{M}(G')$, so wäre $(M'_{\max} \setminus \{v_M\}) \cup M \supset M_{\max}$ ein echter Modul in G, Widerspruch!

Fall 2. $M' \in \mathcal{M}(G) \setminus \{M_{\max}\}$.
Wir nehmen wiederum an, daß M' nicht maximal sei, d.h. $M' \subset M'_{\max}$ für ein $M'_{\max} \in \mathcal{M}(G')$.

Fall a. $v_M \in M'_{\max}$.
Dann ist $(M'_{\max} \setminus \{v_M\}) \cup M \supset M'$ ein echter Modul in G, Widerspruch!

Fall b. $v_M \notin M'_{\max}$.
Da

$$v_M \bowtie_{G'} M'_{\max} \Longleftrightarrow M \bowtie_G M'_{\max} \quad \text{(bzw. } v_M \,||_{G'}\, M'_{\max} \Longleftrightarrow M \,||_G\, M'_{\max}\text{)}$$

ist $M'_{\max} \supset M'$ ein echter Modul in G, Widerspruch!

Damit ist $\mathcal{M}' \subseteq \mathcal{M}(G')$ bewiesen. $\mathcal{M}'$ ist eine Partition von $V(G')$, da $\mathcal{M}(G)$ wegen $G \in \mathbf{M}$ eine Partition von $V(G)$ ist. Hieraus folgt dann $\mathcal{M}' = \mathcal{M}(G')$ (denn wäre $G' \notin \mathbf{M}$, so wären je zwei echte maximale Module in G' nach Korollar 4.3.8 und Lemma 4.2.2 (2) nicht disjunkt).

Da $K_3 \notin \mathbf{M}$ und $K_2 \in \mathbf{M}$ ist die Aussage „$G' \in \mathbf{M} \Longrightarrow G \in \mathbf{M}$“ nicht allgemeingültig. ■

Satz 4.4.5
Es sei **G** *eine Graphenklasse, die bezüglich Primgraphbildung abgeschlossen ist. Dann gilt:*

(1) Ob ein Graph G in $\mathrm{MExt}^*(\mathbf{G})$ ist, läßt sich sequentiell in derselben Zeit entscheiden, wie die Frage, ob G in **G** *ist.*

(2) Läßt sich auf einer **CRCW–PRAM** *in Zeit $O(t)$ mit $O(p)$ Prozessoren entscheiden, ob ein Graph $G = (V, E)$ in* **G** *ist, so ist in $O(t + \log^2 |V|)$ Zeit mit $O(p + |V| + |E|)$ Prozessoren entscheidbar, ob G in $\mathrm{MExt}^*(\mathbf{G})$ ist.*

Beweis. Der Graph $\mathrm{Prim}(G)$ kann nach Satz 4.4.3 in der gleichen Komplexität berechnet werden, wie die Berechnung des modularen Baumes für G aus Satz 4.2.1. Mit Lemma 4.4.4 und Induktion folgt

$$G \in \mathrm{MExt}^*(\mathbf{G}) \Longleftrightarrow \mathrm{Prim}(G) \in \mathbf{G}.$$

Hieraus folgt sofort die Behauptung. ■

Es sei **G** eine Klasse von Graphen. Dann bezeichnen wir mit $\mathrm{hered}(\mathbf{G})$ die folgende Graphenklasse: $G \in \mathrm{hered}(\mathbf{G})$ genau dann, wenn jeder induzierte Teilgraph in G aus **G** ist. Falls **G** eine Klasse von zusammenhängenden Graphen ist, so fordert man natürlich nur, daß jeder zusammenhängende Teilgraph in G wieder aus **G** ist. **G** heißt eine *hereditäre Graphenklasse*, falls $\mathbf{G} = \mathrm{hered}(\mathbf{G})$ gilt.

Ein Knoten v aus G heißt *extremal*, falls v einen *Maximum–Nachbar* besitzt, d.h. es gibt einen Knoten $w \in N[v]$ so, daß $D(v, 2) = N[w]$ gilt. Ein Knotenreihenfolge $(v_1, \ldots, v_n)$ von V heißt eine *Maximum–Nachbarschaftsordnung*, wenn der Knoten v_i extremal in $G_i := G(\{v_i, \ldots, v_n\})$ ist. Einen Maximum–Nachbarn von v_i (in G_i) bezeichnen wir mit $\mathrm{mn}_{G_i}(v_i)$. Diejenigen Graphen, die eine Maximum–Nachbarschaftsordnung besitzen, nennt man *dual chordale* Graphen (**dual chordal**). Die dual chordalen Graphen wurden in [BDCV93] eingeführt und charakterisiert.

Satz 4.4.6
Die folgenden Graphenklassen sind bezüglich Primgraphbildung abgeschlossen:

(1) jede hereditäre Graphenklasse (d.h. insbesondere Bäume und Wälder),

(2) dual chordale Graphen,

(3) homogen geordnete Graphen (Definition siehe [BDN94]).

Beweis. (1) ist trivial, da $\mathrm{Prim}(G)$ ein induzierter Teilgraph in G ist, und $\mathrm{Prim}(G)$ genau dann zusammenhängend ist, wenn G zusammenhängend ist.

Es sei $G \neq K_1$ ein dual chordaler Graph mit der Maximum–Nachbarschaftsordnung σ. Weiter sei M ein (echter) Modul in G sowie $v_M \in M$. Wir zeigen nun, daß auch der Graph $G' := \mathrm{MRed}(G, M, v_M)$ dual chordal ist. Dazu sei σ' die folgende Eliminationsordnung der Knoten aus G':

> Es sei v_r^M der in σ am weitesten rechts stehende Knoten aus M. Wir entfernen nun in σ alle Knoten aus $M \setminus \{v_r^M\}$ und belassen die übrigen Knoten in ihrer (bzgl. σ) ursprünglichen Reihenfolge. Anschließend ersetzen wir den Knoten v_r^M durch v_M. Die erhaltene Eliminationsordnung sei σ'.

Wir zeigen nun, daß $\sigma' =: (v_1, \ldots, v_k)$ eine Maximum–Nachbarschaftsordnung in G' ist.

Ist i die Position von v_M in σ', so folgt aus der Konstruktion von σ', und da M ein Modul in G ist, daß $(v_i, \ldots, v_k)$ eine Maximum–Nachbarschaftsordnung in $G'(\{v_i, \ldots, v_k\})$ ist. Sei nun $l \in \{1, \ldots, i-1\}$. Ist $\mathrm{mn}_{G_j}(v_l) \notin M$ (j bezeichne die Position von v_l in σ), so ist $\mathrm{mn}_{G_j}(v_l)$ auch ein Maximum–Nachbar von v_l in G'_l. Andernfalls, d.h. falls $\mathrm{mn}_{G_j}(v_l) \in M$, ist wegen der Modularität von M der Knoten v_M ein Maximum–Nachbar von v_l in G'_l.

(3) wurde bereits in [BDN94] (Corollary 4.5) bewiesen. ■

Korollar 4.4.7
Die Graphen aus MExt^*(**Wälder**)*,* MExt^*(**Bäume**) *und* MExt^*(**dual chordal**) *können in Linearzeit erkannt werden.*

Beweis. Bäume und Wälder lassen sich mit Breitensuche (**BFS**) oder Tiefensuche (**DFS**) in Linearzeit erkennen ([Bra94]). In [BDCV93] wird gezeigt, daß auch dual chordale Graphen in Linearzeit erkannt werden können. Somit folgt die Behauptung aus den Sätzen 4.4.5 und 4.4.6. ■

Auch die hereditäre Klasse von $\mathrm{MExt}^*(\mathbf{G})$ ist oftmals interessant. Wir setzen abkürzend

$$\mathrm{HMExt}^*(\mathbf{G}) := \mathrm{hered}(\mathrm{MExt}^*(\mathbf{G})).$$

Folgende Eigenschaften sieht man sofort:

Lemma 4.4.8
Es sei **G** *eine hereditäre Graphenklasse. Dann gelten folgende Aussagen:*

(1) $\mathbf{G} \subseteq \mathrm{HMExt}^*(\mathbf{G})$.

(2) $G \in \mathrm{HMExt}^*(\mathbf{G})$ *genau dann, wenn jeder induzierte Teilgraph (bzw. zusammenhängende induzierte Teilgraph, falls* **G** *eine Klasse von zusammenhängenden Graphen ist) von* G *entweder einen nichttrivialen Modul enthält oder aus* **G** *ist.* ■

Für die Klasse HMExt^*(**chordal**) ist bereits eine Charakterisierung durch verbotene Teilgraphen bekannt.

Satz 4.4.9 ([Ola89])
Ein Graph G *ist genau dann aus* HMExt^*(**chordal**)*, wenn er keinen* C_n*,* $n \geq 5$*, sowie keinen Graphen aus Abbildung 4.2 als induzierten Teilgraphen enthält (sogenannte* Weak bipolarizable graphs*).* ■

Ein Graph G heißt *Cograph*, wenn G keinen P_4 als induzierten Teilgraphen besitzt.

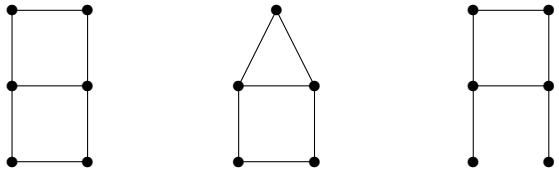
Lemma 4.4.10 ([Sei74])
Sei G *ein Graph mit mehr als einem Knoten. Ist dann sowohl* G *als auch* $\overline{G}$ *zusammenhängend, so ist* G *kein Cograph.* ■

Korollar 4.4.11
Die Klasse $\mathrm{HMExt}^*(\{K_1, K_2\})$ *ist gleich der Klasse der zusammenhängenden Cographen.*

Beweis. Die Inklusion „$\subseteq$“ ist klar, da ein P_4 ein Primgraph ist.

Die umgekehrte Inklusion sieht man wie folgt: Sei $G \neq K_1$ ein zusammenhängender Cograph. Dann ist nach Lemma 4.4.10 das Komplement $\overline{G}$ von G nicht zusammenhängend. Aus dem Beweis von Satz 4.4.3 folgt $\mathrm{Prim}(G) = K_2$ und somit $G \in \mathrm{MExt}^*(\{K_1, K_2\})$. Da die Klasse der Cographen eine hereditäre Graphenklasse ist, folgt $G \in \mathrm{HMExt}^*(\{K_1, K_2\})$. ■

Abbildung 4.2: Ein Domino, Haus und ein „A“



Domino Haus „A“

Es sei $G \in \mathrm{MExt}^*(\textbf{Bäume})$. Weiter seien $T = (V, E)$ ein Baum und $(M_v)_{v \in V}$ disjunkte Moduln in G. $(T, (M_v)_{v \in V})$ heißt eine *Baumdarstellung* von G, wenn

$$G = \mathrm{MExt}(\mathrm{MExt}(\ldots \mathrm{MExt}(T, v_n, M_{v_n}), \ldots, v_2, M_{v_2}), v_1, M_{v_1}),$$

wobei $V = \{v_1, \ldots, v_n\}$. T nennen wir den *unterliegenden Baum* der Baumdarstellung $(T, (M_v)_{v \in V})$.

Aus dem Beweis von Satz 4.4.3 folgt sofort

Korollar 4.4.12
Sei $G \in \mathrm{MExt}^*(\textbf{Bäume})$. Dann besitzt G eine Baumdarstellung. Sie kann sequentiell und parallel in der gleichen Komplexität berechnet werden, wie die Berechnung des modularen Baumes für G aus Satz 4.2.1. ■

4.5 Das RDS–Problem auf $\mathrm{MExt}^*(\textbf{Bäume})$

Wir wissen bereits, daß jeder Graph G aus $\mathrm{MExt}^*(\textbf{Bäume})$ eine Baumdarstellung besitzt (Korollar 4.4.12). Diese Baumstruktur werden wir zur Lösung des **RDS**–Problems auf G benutzen.

Die Module der Baumdarstellung können jedoch einen beliebigen Graphen in G induzieren, was ungünstig ist. Daher untersuchen wir im Abschnitt 5.1 zunächst, ob wir uns auf spezielle Module beschränken dürfen. Wir erhalten (Satz 4.5.2): Beim **RDS**–Problem auf $\mathrm{MExt}^*(\textbf{Bäume})$ können wir uns auf Graphen $G \in \mathrm{MExt}^*(\textbf{Bäume})$ in Baumdarstellung beschränken, so daß für jeden Modul M der Baumdarstellung gilt: $G(M)$ ist entweder ein K_1 (d.h. M ist einelementig) oder ein $2K_1$ (d.h. M besteht aus zwei nichtadjazenten Knoten), dessen beide Knoten den r–Wert 1 besitzen.

Im Abschnitt 5.3 führen wir eine spezielle Eliminationsordnung für Wurzelbäume ein (sog. starke Blätter–Eliminationsordnungen). Diese benötigen wir für den unterliegenden Baum der Baumdarstellung von G im Abschnitt 5.4, wo wir einen Algorithmus entwickeln, der das **RDS**–Problem auf $\mathrm{MExt}^*(\textbf{Bäume})$ in Linearzeit löst.

4.5.1 Einschränkung des Problems auf spezielle Baumdarstellungen

Es sei (G, r) ein *knotenbewerteter Graph*, d.h. G ist ein Graph und $r : V(G) \to \mathbb{N}$ eine zugehörige Knotenbewertung.

Jeden Modul M aus G zerlegen wir wie folgt in drei paarweise disjunkte Teilmengen: $M_0 := \{v \in M : r(v) = 0\}$, $M_1 := \{v \in M : r(v) = 1\}$ und $M_{\geq 2} := \{v \in M : r(v) \geq 2\}$.

Weiter definieren wir vier verschiedene Typen von speziellen Moduln in einem knotenbewerteten Graphen:

Typ	Modul M	G(M)	r–Werte
1	$\{v_1\}$	K_1	$r(v_1) \in \mathbb{N}$
2	$\{v_1, v_2\}$	$2K_1$	$r(v_1) = r(v_2) = 1$
$3a$	$\{v_1, v_2\}$	$2K_1$	$r(v_1) = 0, r(v_2) = 1$
$3b$	$\{v_1, v_2, v_3\}$	$3K_1$	$r(v_1) = 0, r(v_2) = r(v_3) = 1$

Entsprechend nennen wir für $x \in \{1, 2, 3a, 3b\}$ einen knotenbewerteten Graphen (H, s) vom Typ x, wenn $V(H)$ ein Modul vom Typ x in H ist.

Satz 4.5.1
Es sei **G** *eine Klasse von zusammenhängenden Graphen. Läßt sich das* **RDS***–Problem für folgende Instanzen (G, r) auf* **G** *in Zeit $O(t)$ lösen, so läßt es sich bereits für alle Instanzen auf* **G** *in derselben Zeit lösen.*

- *Entweder ist $G \in$* **M** *und jeder echte Modul in G ist ein Modul vom Typ 1, 2, $3a$ oder $3b$,*
- *oder $G \notin$* **M** *und G besitzt eine join–Zerlegung $V = M_1 \bowtie M_2$, so daß M_1, M_2 Module vom Typ 1, 2, $3a$ oder $3b$ sind.*

Beweis. Es sei M ein beliebiger echter Modul in G. Wir konstruieren nun wie folgt aus G einen knotenbewerteten Graphen (G', r'): Wir ersetzen (in G) „in natürlicher Weise“ den Modul M durch einen Modul M' vom Typ x, $x \in \{1, 2, 3a, 3b\}$, exakter: Sei $v_M \in M$ und (H, s) sei ein knotenbewerteter Graph vom Typ x mit $V(H) = M'$. Wir bilden dann den Graphen $G' := \mathrm{MExt}(\mathrm{MRed}(G, M, v_M), v_M, H)$ und definieren r' durch

$$r'(v) := \begin{cases} r(v), & \text{falls } v \in V \setminus M, \\ s(v), & \text{falls } v \in M'. \end{cases}$$

Weiter geben wir an, wie man aus einer minimalen r'–dominierenden Menge D' in G' eine minimale r–dominierende Menge D in G berechnen kann.

Für den Beweis der Korrektheit benützen wir folgende Aussagen, die leicht einzusehen sind:

(1) $V(G') = (V(G) \setminus M) \cup M'$,

(2) $d_G(x, y) = d_{G'}(x, y)$ für alle $x, y \in V(G) \setminus M$,

(3) $d_G(x, M) = d_{G'}(x, M')$ für alle $x \in V(G) \setminus M$,

(4) $d_G(m_1, m_2) \leq 2$ für alle $m_1, m_2 \in M$.

Wir initialisieren D zunächst durch $D := D' \cap (V(G) \setminus M) = D' \setminus M'$.

Fall 1a. $M_0 = \emptyset$ und es existiert ein $x \in M$ mit $M_1 \subseteq N_G[x]$.

Dies ist insbesondere dann der Fall, wenn $M_0 = M_1 = \emptyset$. Wir ersetzen M durch den Modul $M' := \{v_M\}$ vom Typ 1 mit $r'(v_M) := \min_{x \in M} r(x) \geq 1$. Falls $v_M \in D'$, so nehmen wir x in D auf.

Korrektheit: Zunächst zeigen wir, daß D eine r–dominierende Menge in G ist. Ist $v_M \notin D'$, so r–dominiert D nach (2) alle Knoten aus $V \setminus M$. Da v_M in G' durch D' r'–dominiert wird, wird M nach (3) durch D r–dominiert. Ist $v_M \in D'$, so r–dominiert D nach (2) und (3) alle Knoten aus $V \setminus M$ und wegen $x \in D$ auch alle Knoten aus M (x r–dominiert alle Knoten aus M nach Voraussetzung und (4)).

Sei nun $D_{\min}$ eine minimale r–dominierende Menge in G. Dann gilt $|D_{\min} \cap M| \leq 1$, da andernfalls $(D_{\min} \setminus M) \cup \{x\}$ eine echt kleinere r–dominierende Menge in G wäre. Ist $|D_{\min} \cap M| = 0$ (bzw. $|D_{\min} \cap M| = 1$), so ist $D_{\min}$ (bzw. $D_{\min} \cup \{v_M\}$) eine r'–dominierende Menge in G'.

Damit ist D wegen $|D| = |D'|$ tatsächlich eine minimale r–dominierende Menge in G.

Fall 1b. $M_0 \neq \emptyset$ und M_0 r–dominiert M in G.

Wir ersetzen M durch den Modul $M' := \{v_M\}$ vom Typ 1 mit $r'(v_M) := 0$ und fügen M_0 zu D hinzu.

Die Korrektheit beweist man wie im Fall 1a. Man beachte, daß wegen $r'(v_M) := 0$ der Knoten v_M in D' enthalten sein muß.

Fall 2. $M_0 = \emptyset$ und es existiert kein $x \in M$ mit $M_1 \subseteq N_G[x]$ ($\Longrightarrow M_1 \neq \emptyset$).

Dann ersetzen wir M durch den Modul $M' = \{v_M^1, v_M^2\}$ vom Typ 2. Falls $|D' \cap M'| = 1$ nehmen wir einen beliebigen Knoten aus M zu D hinzu. Falls $|D' \cap M'| = 2$ nehmen wir einen beliebigen Knoten aus M und einen beliebigen Knoten aus $N(M)$ ($N(M) \neq \emptyset$, da G zusammenhängend ist) zu D hinzu.

Korrektheit: Ist $|D' \cap M'| = 1$, so ist $D' \cap N_{G'}(M') \neq \emptyset$ (und damit $D \cap N_G(M) \neq \emptyset$), da der Knoten aus $D' \cap M'$ (in G') den anderen Knoten aus D' nicht r'–dominieren kann. Somit ist klar (man beachte (2)–(4)), daß D in diesem Fall eine r–dominierende Menge in G ist.

Ist $|D' \cap M'| = 2$, so r–dominiert D den Graphen G, da der Knoten aus $N(M) \cap D$ die gesamte Menge M (in G) r–dominiert.

Sei nun $D_{\min}$ eine minimale r–dominierende Menge in G. Dann gilt $|D_{\min} \cap M| \leq 2$, da andernfalls $(D_{\min} \setminus M) \cup \{x, m\}$, $m \in M$, $x \in N(M)$, eine echt kleinere r–dominierende Menge in G wäre. Ist $|D_{\min} \cap M| = 0$, so ist $D_{\min}$ auch eine r'–dominierende Menge in G'. Ist $|D_{\min} \cap M| = 1$ (bzw. $|D_{\min} \cap M| = 2$), so ist $(D_{\min} \setminus M) \cup \{v_M^1\}$ (bzw. $(D_{\min} \setminus M) \cup \{v_M^1, v_M^2\}$) eine r'–dominierende Menge in G'.

Damit ist D tatsächlich eine minimale r–dominierende Menge in G (da $|D| = |D'|$).

Fall 3. $M_0 \neq \emptyset$ und M_0 r–dominiert nicht M in G.

Dann sei M_1' die Menge aller Knoten aus M, die nicht durch einen Knoten aus M_0 r–dominiert werden, d.h.

$$M_1' := \{x \in M_1 : x \notin N(M_0)\}.$$

Fall 3a. Es existiert ein $x \in M$, das ganz M_1' in G r–dominiert.
Dann ersetzen wir M durch den Modul $M' := \{v_M^1, v_M^2\}$, $r(v_M^1) = 0$, $r(v_M^2) = 1$ vom Typ $3a$. Wir nehmen M_0 zu D hinzu. Falls $v_M^2 \in D'$, so nehmen wir auch x zu D hinzu.

Die Korrektheit beweist man wie im Fall 1a.

Fall 3b. Es existiert kein $x \in M$, das ganz M_1' in G r–dominiert.
Dann ersetzen wir M durch den Modul $M' := \{v_M^1, v_M^2, v_M^3\}$, $r(v_M^1) = 0$, $r(v_M^2) = r(v_M^3) = 1$ vom Typ $3b$. Wir nehmen M_0 zu D hinzu.
Man beachte, daß $D' \cap \{v_M^2, v_M^3\} = \emptyset$, denn wäre $i := |D' \cap \{v_M^2, v_M^3\}| \in \{1, 2\}$ so erhalten wir wie folgt einen Widerspruch:

1. $i = 1$.
   Dann ist wegen $v_M^1 \in D'$ auch $D' \setminus \{v_M^2, v_M^3\}$ eine r'–dominierende Menge in G' mit weniger Elementen als D' (beachte: $D' \cap N_{G'}(M') \neq \emptyset$).
2. $i = 2$.
   Es sei x ein beliebiger Knoten aus $N_{G'}(M')$. Dann r'–dominiert auch $(D' \setminus \{v_M^2, v_M^3\}) \cup \{x\}$ den Graphen G', Widerspruch!

Die Korrektheit beweist man wie im Fall 2.

Die Korrektheit folgt nun induktiv, indem man im Falle $G \in \mathbf{M}$ jedes $M \in \mathcal{M}(G)$ bzw. im Falle $G \notin \mathbf{M}$ die beiden Module einer beliebigen join–Zerlegung derart konvertiert.

Nun zur Komplexität: Wir zerlegen M zunächst in M_0, M_1, $M_{\geq 2}$ und berechnen die Menge A der Knoten aus M, die M_0 in G r–dominiert. Weiter sei $B := M_1 \setminus A$. Ist $B \neq \emptyset$, so prüfen wir, ob es einen Knoten $v \in M$ gibt, der B in G r–dominiert, d.h. man muß prüfen, ob $M \cap \bigcap_{b \in B} N_G[b] \neq \emptyset$ ist. Dies geht in Zeit $O(|M| + \sum_{b \in B} \deg(b))$, indem man die Nachbarschaftslisten der Knoten aus B durchläuft und für jeden Knoten aus M die Anzahl der Nachbarn in B berechnet. ■

Ein Beispiel zu der im letzten Beweis beschriebenen Konvertierung, findet man in Abbildung 4.3 (ein knotenbewerteter Graph aus MExt^*(**Bäume**) in Baumdarstellung) und Abbildung 4.4 (der gemäß Satz 4.5.1 konvertierte Graph aus Abbildung 4.3).

Wir spezialisieren nun den letzten Satz für die Graphenklasse MExt^*(**Bäume**). Hierdurch erhalten wir eine weitere Vereinfachung.

Satz 4.5.2
Beim **RDS**–*Problem auf* MExt^*(**Bäume**) *können wir uns auf Graphen* $G \in \mathrm{MExt}^*$(**Bäume**) *beschränken, die in einer Baumdarstellung* $(T, (M_v)_{v \in V(T)})$ *gegeben sind, so daß für jedes* $v \in V(T)$ *die Menge* M_v *ein Modul vom Typ 1 oder 2 in* G *ist.*

Beweis. Es sei $G \in \mathrm{MExt}^*$(**Bäume**), $r : V(G) \to \mathbb{N}$ eine Knotenbewertung. Eine Baumdarstellung $(T, (M_v)_{v \in V(T)})$ von G kann nach Korollar 4.4.12 in Linearzeit bestimmt werden. Weiter können wir nach Satz 4.5.1 annehmen, daß jedes M_v, $v \in V(T)$, in G ein Modul des Typs 1, 2, $3a$ oder $3b$ ist.

Wir modifizieren nun die Baumdarstellung $(T, (M_v)_{v \in V(T)})$ wie folgt in eine Baumdarstellung $(T', (M_v')_{v \in V(T')})$ eines Graphen $G' \in \mathrm{MExt}^*$(**Bäume**): Sei v ein Knoten in T, so daß $M := M_v$ in G ein Modul vom Typ 3 (= Typ $3a$ oder Typ $3b$) ist, d.h. M enthält genau einen Knoten v_0 mit $r(v_0) = 0$ und einen oder zwei Knoten mit r–Wert 1. Weiter seien $u_1, \ldots, u_k$ die

Abbildung 4.3: Ein knotenbewerteter Graph aus MExt^*(**Bäume**) in Baumdarstellung

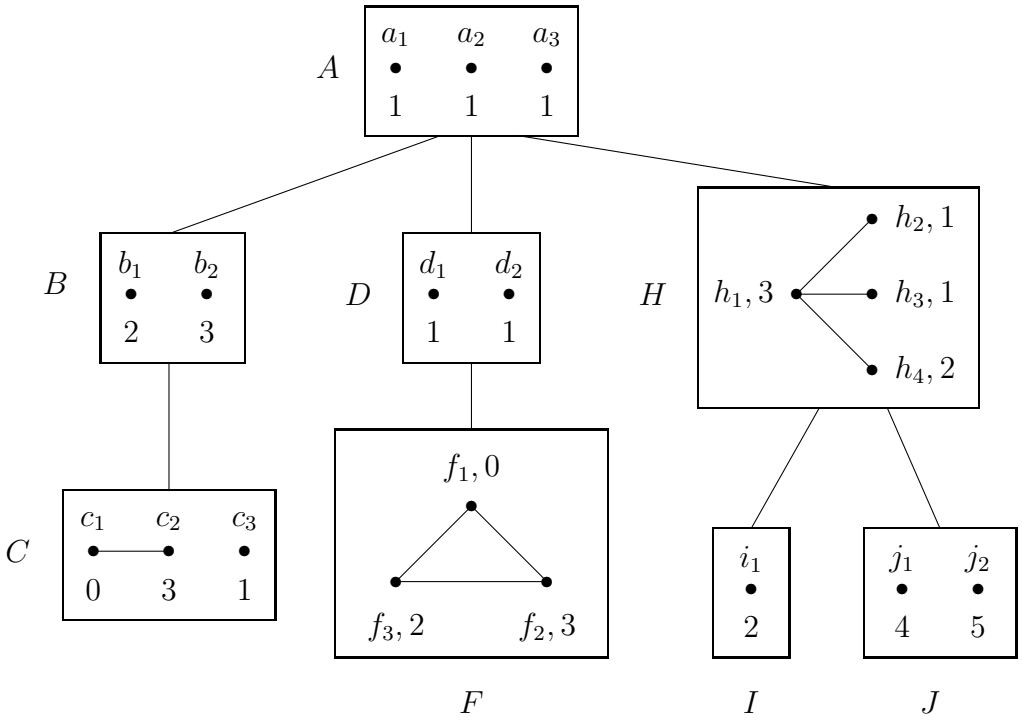


Abbildung 4.4: Der gemäß Satz 4.5.1 konvertierte Graph aus Abbildung 4.3

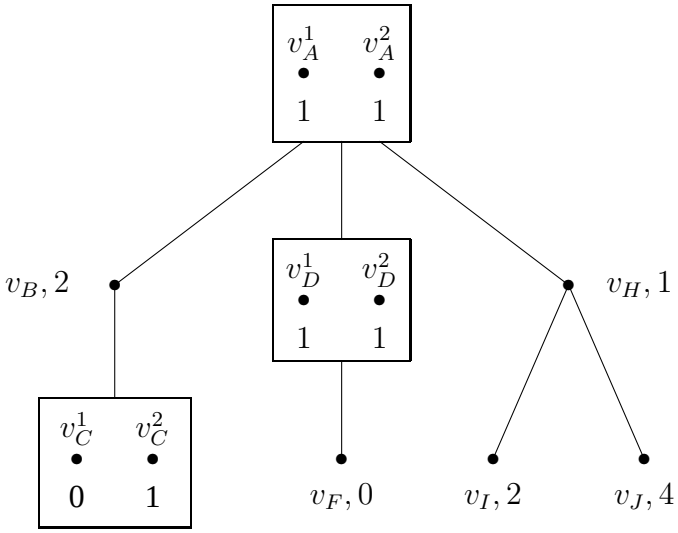
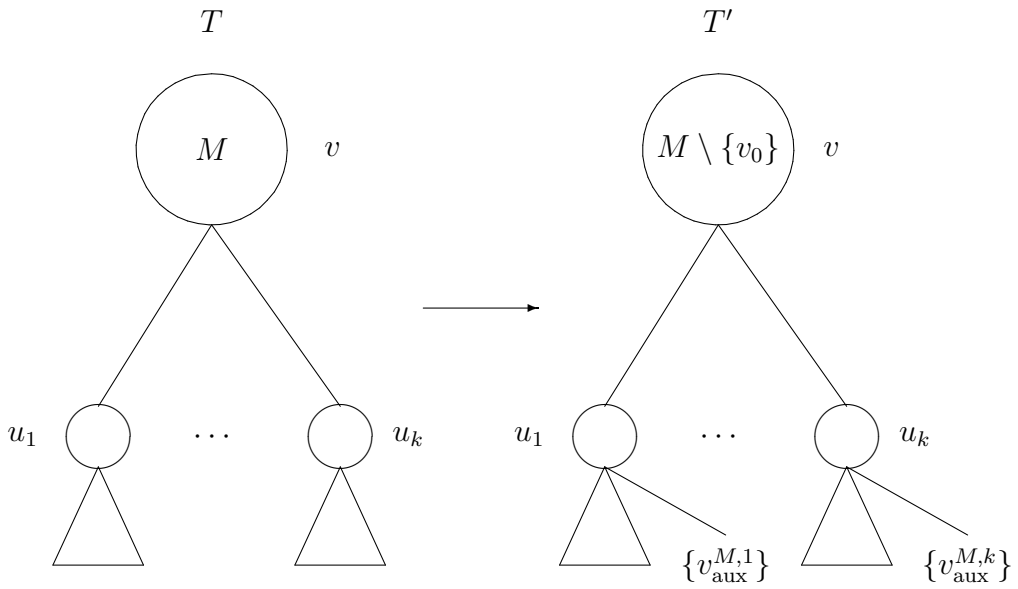


Abbildung 4.5: Skizze zur Konvertierung in Satz 4.5.2



Nachbarn von v in T. Wir entfernen nun v_0 aus M und hängen an jeden Knoten u_i, $i = 1, \ldots, k$, (in T) einen neuen Knoten als Blatt an, der einen Modul $\{v_{\text{aux}}^{M,i}\}$ vom Typ 1 mit r–Wert 0 repräsentiert, vgl. Abbildung 4.5.

Dann gilt: Für jeden Knoten $x \neq v_0$ in G ist

$$\min_{i=1}^{k} d_{G'}(x, v_{\text{aux}}^{M,i}) = d_G(x, v_0),$$

d.h. $\{v_{\text{aux}}^{M,i} : i = 1, \ldots, k\}$ dominiert in G' genau diejenigen Knoten, die v_0 in G dominiert.

Hieraus folgt: Ist D' eine minimale r–dominierende Menge in G', so ist eine minimale r–dominierende Menge in G gegeben durch

$$(D' \setminus \{v_{\text{aux}}^{M,i} : i = 1, \ldots, k\}) \cup \{v_0\}.$$

Auf diese Weise lassen sich alle Module des Typs 3 in Module der Typen 1 und 2 konvertieren. Da die Anzahl der insgesamt neu hinzugefügten Knoten in $O(\sum_{v \in V(T)} \deg(v)) = O(2|E(T)|)$ liegt, ist der Satz bewiesen. ■

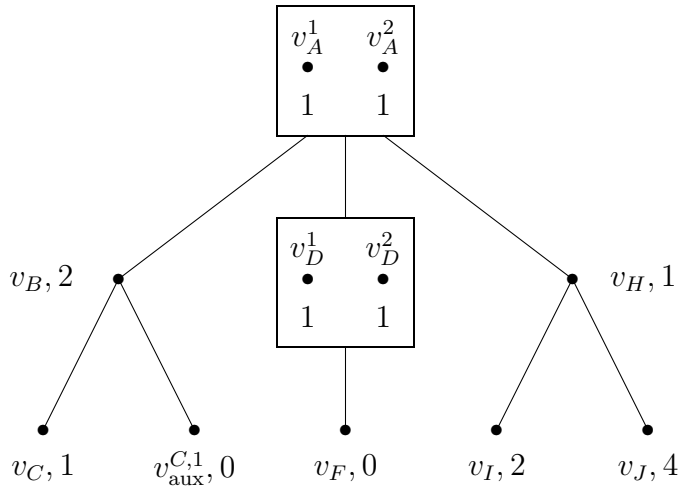
4.5.2 Starke Blätter–Eliminationsordnungen

Sei $T = (V, E)$ ein Wurzelbaum und $v \in V$. Der Knoten v heißt ein *Vater von Blättern*, falls gilt:

- $\text{outdeg}(v) \geq 1$, d.h. v besitzt mindestens ein Kind,
- alle Kinder von v sind Blätter.

$\sigma = ((v_1, B_1), \ldots, (v_{k-1}, B_{k-1}), v_k)$ heißt eine *starke Blätter–Eliminationsordnung* in T genau dann, wenn

Abbildung 4.6: Der gemäß Satz 4.5.2 konvertierte Graph aus Abbildung 4.4



- für $i = 1, \ldots, k-1$ ist v_i ein Vater von Blättern in $T_i := T - \bigcup_{j=1}^{i-1} B_j$ und B_i sind die Kinder von v_i in T_i,
- $V \setminus \bigcup_{j=1}^{k-1} B_j = \{v_k\}$ (dann ist natürlich $v_k = v_{k-1}$).

Die Bezeichnung „starke Blätter–Eliminationsordnung“ ist hierbei wie folgt motiviert: Ist σ_i eine beliebige Anordnung der Knoten aus B_i, $i = 1, \ldots, k-1$, so ist $(\sigma_1, \ldots, \sigma_{k-1}, v_k)$ eine Blätter–Eliminationsordnung in T.

Satz 4.5.3
Jeder Wurzelbaum $T_w = (V, E)$ besitzt eine starke Blätter–Eliminationsordnung σ und sie kann in Linearzeit berechnet werden.

Beweis. Die Existenz sieht man leicht: Ist $T \neq K_1$ und $v \in V$ ein Knoten mit $\mathrm{depth}(v) = \mathrm{depth}(T)$, so ist v ein Blatt und $\mathrm{parent}(v)$ ein Vater von Blättern in T. Hieraus folgt induktiv die Existenz einer starken Blätter–Eliminationsordnung. Der letzte Knoten von σ ist dabei immer die Wurzel w.

Nun zum Beweis, daß σ in Linearzeit berechnet werden kann. Wir bestimmen zunächst für jeden Knoten v dessen Ausgangsgrad $\mathrm{outdeg}(v)$. Dann gilt für $v \neq w$:

$$v \text{ ist ein Blatt} \iff \mathrm{outdeg}(v) = 0.$$

Anschließend berechnen wir für jeden Knoten v die Anzahl $\mathrm{Loutdeg}(v)$ der Kinder von v, die Blätter sind. Dann ist v ein Vater von Blättern genau dann, wenn

$$\mathrm{outdeg}(v) \neq 0 \text{ und } \mathrm{Loutdeg}(v) = \mathrm{outdeg}(v).$$

Die starke Blätter–Eliminationsordnung berechnet man dann mit der üblichen Warteschlangentechnik[4], d.h. man geht wie folgt vor:

(1) Sei $W = \emptyset$ eine zunächst leere Warteschlange und $\sigma := (\,)$.

[4]Ein klassisches Beispiel, wo die Warteschlangentechnik verwendet wird, ist Breitensuche ([Bra94]).

(2) Füge alle Knoten v mit $\mathrm{outdeg}(v) \neq 0$ und $\mathrm{Loutdeg}(v) = \mathrm{outdeg}(v)$ zu W hinzu.

(3) Solange $W \neq \emptyset$ führe folgendes aus:

(a) Sei $x \in W$ beliebig. Lösche x aus W.

(b) Entferne die Kinder K_x von x aus dem aktuellen Baum.

(c) Füge (x, K_x) am Ende von σ hinzu.

(d) Sei $y := \mathrm{parent}(x)$. Aktualisiere

$$\mathrm{Loutdeg}(y) := \mathrm{Loutdeg}(y) + 1.$$

Ist $\mathrm{Loutdeg}(y) = \mathrm{outdeg}(y)$, so füge y zu W hinzu.

(4) Der aktuelle Baum besteht dann nur noch aus einem einzelnen Knoten v. Füge v ans Ende von σ hinzu. ■

4.5.3 Der Algorithmus

Um das **RDS**–Problem für einen Graphen $G \in \mathrm{MExt}^*(\textbf{Bäume})$, $r : V(G) \to \mathbb{N}$ eine Knotenbewertung, zu lösen, dürfen wir nach Abschnitt 4.5.1 annehmen, daß G in einer Baumdarstellung $(T = (V, E), (M_v)_{v \in V})$ gegeben ist und daß jedes M_v, $v \in V$, ein Modul vom Typ 1 oder 2 (in G) ist. Weiter nehmen wir an, daß T als Wurzelbaum vorliegt.

Um die Bezeichnungsweise zu vereinfachen, führen wir folgende Sprechweisen ein: Einen Knoten $v \in V$ nennen wir einen *i–Knoten*, falls M_v (in G) ein Modul vom Typ i ist, $i = 1, 2$. Weiter identifizieren wir v mit dem Knoten aus M_v, falls v ein 1–Knoten ist. Ist v ein 2–Knoten, so sei $\{v_1, v_2\} := M_v$. Für M_v schreiben wir auch v. Sprechen wir z.B. von den Knoten im Teilbaum T_x, so meinen wir die Knoten aus den zugehörigen Moduln der Knoten aus dem Teilbaum T_x, hierfür schreiben wir auch kurz $V_G(T_x)$, d.h. $V_G(T_x) := \bigcup_{v \in V(T_x)} M_v$. Anstatt r–dominiert schreiben wir häufig auch kurz dominiert.

Der Algorithmus basiert auf entsprechenden Algorithmen für das **RDS**–Problem auf Bäumen ([Sla76], [HY90]) und dual chordalen Graphen ([BCD94]).

Wenn wir im folgenden von den Parametern eines Knotens des aktuellen Baumes sprechen, so meinen wir für einen 1–Knoten v die zwei Parameter $c(v)$ und $r(v)$ und für einen 2–Knoten v die fünf Parameter $c(v)$, A_v, B_v, C_v und D_v. Die Bedeutung dieser Parameter beschreiben wir später. Hier wollen wir zunächst nur das grobe Vorgehen des Algorithmus darlegen: Zunächst werden die Parameter der Blätter von T mit Werten belegt, und eine Menge R wird durch $R := \emptyset$ initialisiert. Anschließend führen wir wiederholt den folgenden Schritt aus, bis der aktuelle Baum ein K_1 ist:

> Wir berechnen einen Vater von Blättern x in dem gerade aktuellen Baum. Anschließend löschen wir die Kinder von x und belegen die Parameter von x mit Werten. Diese Werte bestimmen wir hierbei aus den Parametern der Kinder von x. Weiter entscheiden wir, ob wir „neue“ Knoten zur Menge R hinzufügen.

Anschließend, d.h. wenn der aktuelle Baum nur noch aus der Wurzel w besteht, können wir eine minimale r–dominierte Menge für G aus R und den Parametern von w berechnen.

Den im jeden Schritt gerade vorliegenden Baum nennen wir den *aktuellen Baum*. Entsprechend unterscheiden wir zwischen *aktuellen* und *nicht aktuellen* Knoten.

Sei x ein Blatt im gerade aktuellen Baum und $R_x := R \cap V_G(T_x)$. Ist x ein 1–Knoten, so haben die Parameter $r(x)$, $c(x)$ folgende Bedeutung:

- $c(x)$ gibt den minimalen Abstand (in T) von x zu einem Knoten aus R_x an, falls ein solcher existiert. Andernfalls sei $c(x) := \infty$.
- $r(x)$ gibt den maximal möglichen Abstand an, in dem x im aktuellen Baum dominiert werden muß, um alle noch nicht von R_x dominierten nicht aktuellen Knoten in T_x und x zu dominieren.

Ist x ein 2–Knoten, so haben die Parameter von x folgende Bedeutung:

- $c(x)$ gibt, wie für einen 1–Knoten, den minimalen Abstand (in T) von x zu einem Knoten aus R_x an, falls ein solcher existiert. Sonst sei $c(x) := \infty$.
- A_x (bzw. B_x) ist eine bzgl. der (Mengen–)Inklusion minimale Menge, die folgende Eigenschaften erfüllt:
  (1) $A_x \cup R_x$ (bzw. $B_x \cup R_x$) dominiert alle Knoten aus T_x.
  (2) $A_x \cap x \neq \emptyset$ (bzw. $B_x \cap x = \emptyset$).
- C_x (bzw. D_x) ist eine bzgl. der (Mengen–)Inklusion minimale Menge, die folgende Eigenschaften erfüllt:
  (1) $C_x \cup R_x$ (bzw. $D_x \cup R_x$) dominiert alle Knoten aus $T_x - x$ (bzw. alle Knoten y aus $T_x - x$ mit $d_G(x_1, y) \geq r(y)$).[5]
  (2) $C_x \cup R_x$ (bzw. $D_x \cup R_x$) dominiert *nicht* alle Knoten aus $x = \{x_1, x_2\}$.
  (3) $C_x \cap x \neq \emptyset$ (bzw. $D_x \cap x = \emptyset$).

Natürlich brauchen einige der Mengen A_x, B_x, C_x und D_x nicht unbedingt definiert zu sein.

- A_x ist immer definiert, da $V_G(T_x)$ alle Knoten aus T_x dominiert.
- Die Menge B_x ist genau dann definiert, wenn $T_x \neq (\{x\}, \emptyset)$, denn dann dominiert $V_G(T_x - x)$ den gesamten Teilbaum T_x.
- Ist $R_x \cap N_G(x) \neq \emptyset$ (d.h. $c(x) = 1$), so ist C_x nicht definiert, da R_x bereits (alle Knoten aus) x dominiert. Andernfalls ist C_x immer definiert, da die Menge $\{x_1\} \cup \{v \in V_G(T_x) : v \notin N_G[x]\}$ alle Knoten aus $T_x - x$, aber nicht x_2 dominiert.
- Ist $R_x \cap N_G(x) \neq \emptyset$ (d.h. $c(x) = 1$), so ist D_x nicht definiert. Andernfalls ist D_x genau dann definiert, wenn kein Kind y von x ein 1–Knoten mit r–Wert 1 oder ein 2–Knoten ist, welcher zusätzlich ein Blatt in T ist (denn dann dominiert $V_G(T_x) \setminus N_G[x]$ alle Knoten y aus $T_x - x$ mit $d_G(x_1, y) \geq r(y)$, jedoch nicht x).

[5] Es sei angemerkt, daß die Knoten aus $\{y \in V_G(T_x - x) : d_G(x_1, y) \geq r(y)\}$ gerade diejenigen Knoten aus $V_G(T_x - x)$ sind, die nicht durch $\mathrm{parent}(x)$ dominiert werden können (falls x nicht die Wurzel in T ist).

Der folgende Satz untersucht (im Falle ihrer Definiertheit) die Kardinalität der Mengen B_x, C_x und D_x in Bezug auf A_x.

Satz 4.5.4
Es gilt:

(1) Ist B_x definiert, so gilt $|B_x| + 1 \geq |A_x|$,

(2) Ist C_x definiert, so gilt $|C_x| + 1 \geq |A_x|$,

(3) Ist D_x definiert, so gilt $|D_x| + 2 \geq |A_x|$,

(4) Sind B_x und D_x definiert, so gilt $|D_x| + 1 \geq |B_x|$.

Beweis. Wegen der Minimalität von A_x folgt

$$|B_x| + 1 = |B_x \cup \{x_1\}| \geq |A_x|.$$

Ist $T_x = (\{x\}, \emptyset)$, so sind (2) und (3) trivial. Andernfalls sei $v \in N_G(x) \cap V_G(T_x)$, dann gilt

$$|C_x| + 1 = |C_x \cup \{v\}| \geq |A_x|, \qquad |D_x| + 2 = |D_x \cup \{v, x_1\}| \geq |A_x|.$$

(4) beweist man ebenso. ■

Die Mengen B_x, C_x und D_x sind im folgenden meist nur interessant, wenn sie definiert sind und eine kleinere Kardinalität als A_x besitzen. Daher schreiben wir abkürzend: D_x ist *2–optimal*, falls D_x definiert ist und $|D_x| + 2 = |A_x|$ gilt. L heißt *1–optimal*, falls L definiert ist und $|L| + 1 = |A_x|$ erfüllt, $L \in \{B_x, C_x, D_x\}$.

Es seien $P_1, \ldots, P_k$ nicht notwendig definierte Mengen. Dann bezeichne $\min\{P_1, \ldots, P_k\}$ eine definierte Menge aus $\{P_1, \ldots, P_k\}$, die minimale Kardinalität besitzt (falls mind. eine Menge aus $\{P_1, \ldots, P_k\}$ definiert ist). Sind alle Mengen aus $\{P_1, \ldots, P_k\}$ undefiniert, so sei $\min\{P_1, \ldots, P_k\} := \infty$.

Wir kommen nun zur Initialisierung der Parameter der Blätter in T. Da R zu Beginn noch leer ist, müssen wir den c–Wert jedes Blattes mit ∞ initialisieren. Weiter gibt $r(v)$ für ein Blatt v, das ein 1–Knoten ist, dessen r–Wert aus der gegebenen Knotenbewertungsfunktion r auf G an. Die Mengenparameter eines Blattes v, das ein 2–Knoten ist, werden wie folgt initialisiert:

- $A_v = \{v_1, v_2\}$,
- B_v nicht definiert,
- $C_v = \{v_1\}$ und
- $D_v = \emptyset$.

Im Ausgangsgraphen erfüllen die Parameter der Blätter dann trivialerweise die oben angegebene Bedeutung.

Es sei der aktuelle Baum kein K_1 und x sei hierin ein Vater von Blättern. Wir beschreiben nun, wie wir die Parameter von x belegen und welche, wenn überhaupt, Knoten wir zu R hinzufügen. Dabei beweisen wir für jeden Fall, daß die x zugewiesenen Parameter ihrer (oben beschriebenen) Bedeutung gerecht werden, und daß R mit einer minimalen r–dominierenden

Menge für den Restgraphen eine minimale r–dominierende Menge für G liefert (natürlich unter der Voraussetzung, daß die Parameter der Blätter im aktuellen Baum ihrer Bedeutung gerecht werden).

Sei K_x die Menge der Kinder von x im aktuellen Baum. Wir unterscheiden je nach Typ des x–Knotens zwei Fälle:

Fall 1. x ist ein 1–Knoten.

In diesem Fall können wir die Kinder von x sequentiell abpflücken. Hierbei haben die Parameter von x folgende Bedeutung: Sei K^x_{ent} die Menge der schon entfernten Kinder von x und $V_{\text{ent}} := \bigcup_{u \in K^x_{\text{ent}}} V_G(T_u)$.

- $c(x)$ gibt den minimalen Abstand (in T) von x zu einem Knoten aus $R \cap V_{\text{ent}}$ an, falls ein solcher existiert. Andernfalls sei $c(x) := \infty$.
- $r(x)$ gibt den maximal möglichen Abstand an, in dem x dominiert werden muß, um alle noch nicht von $R \cap V_{\text{ent}}$ dominierten Knoten aus $V_{\text{ent}} \cup \{x\}$ zu dominieren.

Klar ist, daß die Parameter von x nach dem Entfernen aller Kinder von x die oben beschriebene Bedeutung erhalten.

Die Parameter von x müssen wir wie folgt initialisieren: $c(x) := \infty$ und $r(x)$ sei der r–Wert des Knotens aus x.

Sei y ein noch nicht entferntes Kind von x und $r(x)$, $c(x)$ erfüllen obige Bedeutung. Wir entfernen dann y und aktualisieren die Parameter von x und R gemäß folgender Fallunterscheidung:

Fall 1.1. y ist ein 1–Knoten.

Fall 1.1.1. $r(y) = 0$.

Da $r(y) = 0$ ist, müssen wir y zur Menge R hinzunehmen, d.h. $R := R \cup \{y\}$. Die Parameter von x werden wie folgt aktualisiert: $c(x) := 1$, da $d(x, y) = 1$. Ist $r(x) \geq 1$, so dominiert $R \cap V_{\text{ent}}$ alle Knoten aus V_{ent} und x. Daher setzen wir $r(x) := \infty$. Ist $r(x) = 0$, so wird $r(x)$ nicht aktualisiert.

Fall 1.1.2. $r(y) > 0$.

Fall 1.1.2.1. ($r(y) = \infty$) oder ($c(x) + 1 \leq r(y)$).

Dann werden alle Knoten in T_y bereits durch die Menge $R \cap V_{\text{ent}}$ dominiert. Daher brauchen wir keinen Knoten aus T_y zu R hinzufügen.[6] Wir aktualisieren $c(x)$ durch $c(x) := \min\{c(x), c(y) + 1\}$. Falls $c(y) + 1 \leq r(x)$, so werden alle Knoten aus $V_{\text{ent}} \cup \{x\}$ bereits durch $R \cap V_{\text{ent}}$ dominiert. Daher setzen wir in diesem Fall $r(x) := \infty$. Falls $c(y) + 1 > r(x)$, so wird x durch keinen Knoten aus R_y r–dominiert. In diesem Fall aktualisieren wir $r(x)$ nicht.

Fall 1.1.2.2. ($r(y) \neq \infty$) und ($c(x) + 1 > r(y)$).

Dann wird y nicht durch $R \cap V_{\text{ent}}$ dominiert. Da $r(y) > 0$ brauchen wir jedoch keinen Knoten aus T_y zu R hinzunehmen.[6] Wir aktualisieren daher den r–Wert von x durch $r(x) := \min\{r(x), r(y) - 1\}$. Weiter sei $c(x) := \min\{c(x), c(y) + 1\}$.

[6] Ist S eine minimale r–dominierende Menge in G mit $R \subseteq S$ und $S \cap (V(T_y) \setminus R) \neq \emptyset$, so ist auch $(S \setminus (V(T_y) \setminus R)) \cup \{x\}$ eine minimale r–dominierende Menge in G.

Fall 1.2. y ist ein 2–Knoten.

Fall 1.2.1. $c(y) = 1$.

Dann sind C_y und D_y undefiniert, da y bereits durch R_y dominiert wird. Um die noch nicht dominierten Knoten aus T_y zu dominieren, muß entweder A_y oder B_y zu R hinzugenommen werden.[7] (Denn: Sei K die Menge der noch nicht durch R dominierten Knoten in T_y. Da kein Knoten aus K durch einen Knoten aus $T - V(T_y)$ dominiert werden kann, muß entweder A_y oder B_y zu R hinzugefügt werden.) Wir unterscheiden zwei Fälle:

Fall 1.2.1.1. B_y ist 1–optimal.

Dann ist es optimal B_y zu R hinzuzunehmen, da $|B_y| + 1 = |A_y|$ und $B_y \cup \{x\}$ mehr Elemente dominiert als A_y. Die Parameter von x aktualisieren wir wie folgt: $c(x) := \min(c(x), c(y) + 1) = \min(c(x), 2)$. Falls $r(x) \geq 2$, so dominiert $R \cap V_{\text{ent}}$ bereits alle Knoten aus $V_{\text{ent}} \cup \{x\}$. Daher setzen wir dann $r(x) := \infty$.

Fall 1.2.1.2. B_y ist nicht 1–optimal.

Hier ist es optimal A_y zu R hinzuzunehmen, da $|B_y| \geq |A_y|$ und A_y einen Knoten aus y enthält. Die Parameter von x aktualisieren wir wie folgt: $c(x) := \min(c(x), 1) = 1$. Falls $r(x) \geq 1$, so dominiert $R \cap V_{\text{ent}}$ bereits alle Knoten aus $V_{\text{ent}} \cup \{x\}$. Daher setzen wir dann $r(x) := \infty$.

Fall 1.2.2. $c(y) > 1$.

Hier werden die Knoten aus y noch nicht durch R_y und damit auch nicht durch R_x dominiert. Wir haben also nur zwei Möglichkeiten, um die noch nicht dominierten Knoten in T_y zu dominieren. Entweder nehmen wir die Knoten aus A_y oder B_y zu R hinzu, so daß dann der gesamte Baum T_y durch R dominiert wird, oder wir nehmen C_y oder D_y zu R hinzu. Dann müssen wir jedoch auch x zu R hinzunehmen, da ein Knoten aus y noch nicht dominiert wird. Um zu entscheiden, welche der beiden Möglichkeiten die Optimalere ist, unterscheiden wir fünf Fälle. Die weiter hinten liegenden Fälle seien dabei mit den vorigen jeweils disjunkt zu verstehen.

Fall 1.2.2.1. D_y ist definiert und 2–optimal.

Dann nehmen wir D_y zu R hinzu und setzen $r(x) := 0$ (d.h. wir fordern, daß x, wenn es entfernt wird, auf jeden Fall zu R hinzugefügt werden muß), sowie

$$c(x) := \begin{cases} \min(c(x), c(y) + 1), & \text{falls } D_y = \emptyset, \\ \min(c(x), c(y) + 1, 3), & \text{falls } D_y \neq \emptyset. \end{cases}$$

Dies ist optimal, da $|D_y \cup \{x\}| \leq |\min\{A_y, B_y, C_y\}|$ und $D_y \cup \{x\}$ mindestens so viele noch nicht dominierte Knoten dominiert wie einer der anderen (definierten) Mengenparameter von y.

Fall 1.2.2.2. B_y ist definiert und 1–optimal.

Dann nehmen wir B_y zu R hinzu und aktualisieren die Parameter von x wie folgt: $c(x) := \min(c(x), 2)$ (da $d(x, B_y) = 2$). Ist $r(x) \geq 2$, so sei $r(x) = \infty$ (Begründung wie im Fall 1.2.1.1).

[7] Man beachte: Werden bereits alle Knoten in T_y durch R_y dominiert, so ist $B_y = \emptyset$.

Warum B_y optimaler ist als A_y und D_y ist klar. Wir müssen also nur bestätigen, daß dies auch dann der Fall ist, wenn C_y 1–optimal ist. Dies sieht man so: Nehmen wir C_y zu R hinzu, so wird ein Knoten aus y noch nicht dominiert. Daher sind wir gezwungen auch x zu R hinzuzunehmen. Da $B_y \cup \{x\}$ genauso viele noch nicht dominierte Knoten dominiert wie $C_y \cup \{x\}$ ist hier B_y eine bessere oder gleichgute Wahl.

Fall 1.2.2.3. C_y ist definiert und 1–optimal.
Dann nehmen wir C_y zu R hinzu und aktualisieren: $c(x) := 1$, $r(x) := 0$.

Fall 1.2.2.4. D_y ist definiert und 1–optimal.
Dann nehmen wir D_y zu R hinzu und aktualisieren: $r(x) := 0$ und

$$c(x) := \begin{cases} \min(c(x), c(y)+1), & \text{falls } D_y = \emptyset, \\ \min(c(x), c(y)+1, 3), & \text{falls } D_y \neq \emptyset. \end{cases}$$

Fall 1.2.2.5. Keine der Mengen–Parameter B_y, C_y und D_y ist optimal.
Dann nehmen wir A_y zu R hinzu und aktualisieren: $c(x) := 1$ und falls $r(x) \geq 1$ setzen wir $r(x) := \infty$.

Fall 2. x ist ein 2–Knoten.

Zunächst entfernen wir sequentiell alle Kinder y von x, die 1–Knoten sind und deren r–Wert ungleich 1 ist. Dazu initialisieren wir $c(x)$ zunächst durch ∞. Ist $r(y) = 0$, so entfernen wir y, fügen y zu R hinzu und setzen $c(x) := 1$. Ist $r(y) > 1$, so entfernen wir y und aktualisieren $c(x)$ durch $c(x) := \min(c(x), c(y)+1)$.

Sind dann bereits alle Kinder abgepflückt, so bekommen die Mengenparameter von x folgende Werte zugewiesen:

Ist $c(x) = 1$, so sind C_x und D_x nicht definiert. $A_x := \{x_1\}$ und $B_x := \emptyset$. Ist $c(x) > 1$, so setzen wir $A_x := \{x_1, y\}$, $B_x := \{y\}$, y ein beliebiges Kind von x,[8] $C_x := \{x_1\}$ und $D_x := \emptyset$ (da $1 = d(x,y) < r(y)$ für alle $y \in K_x$).

Andernfalls gibt es noch Kinder $y_1, \ldots, y_m$ von x, die keine 1–Knoten mit r–Wert ungleich 1 sind.

Dann braucht keiner der schon entfernten Kinder von x zu einem Mengenparameter von x hinzugefügt werden: Sei y ein schon entferntes Kind von x. Ist $r(y) = 0$, so ist dies klar. Andernfalls, d.h. falls $r(y) > 1$, sieht man dies wie folgt: Da $x \cap A_x$, $x \cap C_x \neq \emptyset$ gilt $y \notin A_x$ und $y \notin C_x$. Aus $d(x,y) = 1 < r(y)$ folgt $y \notin D_x$. Da jeder Knoten aus einem noch nicht entfernten Kind, y dominiert, können wir ebenfalls $y \notin B_x$ annehmen.

Die übrig gebliebenen Kinder $y_1, \ldots, y_m$ dürfen wir nicht sequentiell abpflücken. Um die Bezeichnungsweise zu vereinfachen, fassen wir ein Kind y von x, das ein 1–Knoten mit r–Wert 1 ist, als einen 2–Knoten auf, und zwar mit den Mengen $A_y = \{y\}$, $D_y = \emptyset$ (denn: alle Knoten aus T_y, die noch nicht durch R_y dominiert werden, erfüllen $r(y) + d(z,y) \leq r(y)$, d.h. $d(z,y) < r(y)$),

$$B_y := \begin{cases} \{z\}, & \text{falls } N_{T_y}(y) \neq \emptyset,\ z \in N_{T_y}(y), \\ \text{nicht definiert}, & \text{falls } N_{T_y}(y) = \emptyset, \end{cases}$$

[8] Nach Definition besitzt x mindestens ein Kind!

und C_y sei undefiniert.

Die Mengen–Parameter von x werden nun wie folgt aktualisiert:

- Es sei
$$O := \{x_1\} \cup \bigcup_{i=1}^{m} \min\{A_{y_i}, B_{y_i}, C_{y_i}, D_{y_i}\},$$
d.h. O ist diejenige Menge, die x_1 und von jedem Kind y_i, $i = 1, \ldots, m$, eine (bzgl. der Kardinalität) minimale (definierte) Menge enthält. Dann muß $|A_x| \geq |O|$ gelten.
Ist $c(x) = 1$, so können wir $A_x := O$ setzen.
Ist $c(x) > 1$, so muß mindestens von einem Kind y_{i_0} die Menge $A_{y_{i_0}}$ oder $C_{y_{i_0}}$ hinzugenommen werden, damit alle Knoten aus x dominiert werden.[9] Für die übrigen Kinder y_j, $j \neq i_0$, nehmen wir dann eine definierte Menge kleinster Kardinalität hinzu, d.h. also
$$A_x := \{x_1\} \cup K \cup \bigcup_{i \in \{1,\ldots,m\} \setminus \{i_0\}} \min\{A_{y_i}, B_{y_i}, C_{y_i}, D_{y_i}\},$$
wobei
$$K \in \{A_{y_{i_0}}, C_{y_{i_0}}\}.$$
Um i_0 und K anzugeben, unterscheiden wir mehrere Fälle. Die weiter hinten liegenden Fälle seien dabei mit den vorigen jeweils disjunkt zu verstehen.

  Fall I. Es gibt ein $j \in \{1, \ldots, m\}$, so daß die Kardinalität von B_{y_j}, C_{y_j} und D_{y_j} (falls definiert) größer oder gleich der Kardinalität von A_{y_j} ist, d.h. $|A_{y_j}| = |\min\{A_{y_j}, B_{y_j}, C_{y_j}, D_{y_j}\}|$.
  Dann sei $i_0 := j$ und $K := A_{y_{i_0}}$.
  Da $|A_x| = |O|$ ist dies optimal.

  Fall II. Es gibt ein $j \in \{1, \ldots, m\}$, so daß C_{y_j} definiert und 1–optimal ist und D_{y_j} nicht 2–optimal ist.
  Dann sei $i_0 := j$ und $K := C_{y_{i_0}}$.
  In diesem Fall hat C_{y_j} unter den definierten Mengenparameter von y_j die kleinste Kardinalität. Somit gilt wiederum $|A_x| = |O|$.

  Fall III. Jedes Kind y_j besitzt eine optimale Menge als Parameter und falls C_{y_j} definiert und 1–optimal ist, so ist auch D_{y_j} definiert und 2–optimal, $j = 1, \ldots, m$.
  Existiert ein j, so daß C_{y_j} 1–optimal ist, so setzen wir $i_0 := j$ und $K := C_{y_{i_0}}$. Dann gilt $|O| = |A_x| - 1$ (denn nur bei y_j haben wir anstatt der (bzgl. der Kardinalität) optimalen 2–optimalen Menge D_{y_j} die 1–optimale Menge C_{y_j} hinzugenommen). Doch O kann von keinem Kind eine A–Menge (da Fall I nicht erfüllt ist) und keine C–Menge (da Fall II nicht erfüllt ist) enthalten. Somit ist A_x richtig gewählt worden.
  Ist für kein j die Menge C_{y_j} 1–optimal so unterscheiden wir wiederum zwei Fälle:

[9] Wir dürfen o.B.d.A. voraussetzen, daß $|A_x \cap x| = 1$ ist, denn im Falle $|A_x \cap x| = 2$ ist $(A_x \setminus \{x_2\}) \cup \{z\}$, $z \in N_G(x) \cap V_G(T_x)$, genauso optimal.

Fall III.a. Es gibt ein $j \in \{1, \ldots, m\}$, so daß D_{y_j} nicht 2–optimal ist.
Dann sei $i_0 := j$ und $K := A_{y_{i_0}}$.
Die Korrektheit beweist man wie oben, da wieder $|O| = |A_x| - 1$.

Fall III.b. Für jedes Kind y_j ist D_{y_j} 2–optimal.
Dann sei $i_0 \in \{1, \ldots, m\}$ beliebig und $K := A_{y_{i_0}}$.
Da $|O| = |A_x| - 2$ ist, haben wir auch in diesem Fall die richtige Wahl getroffen.

- Ist $c(x) = 1$, so ist $B_x := \bigcup_{i=1}^{m} F_i$, wobei für $i \in \{1, \ldots, m\}$ gilt

$$F_i := \begin{cases} B_{y_i}, & \text{falls } B_{y_i} \text{ definiert und 1–optimal ist,} \\ A_{y_i}, & \text{sonst.} \end{cases}$$

Ist $c(x) > 1$, so muß mindestens von einem Kind y_{i_0} die Menge $A_{y_{i_0}}$ hinzugenommen werden. Wir setzen

$$B_x := A_{y_{i_0}} \cup \bigcup_{i \in \{1,\ldots,m\} \setminus \{i_0\}} \min\{A_{y_i}, B_{y_i}\}.$$

Zur Wahl von i_0 unterscheiden wir:

Fall a. Für ein Kind y_j ist B_{y_j} undefiniert oder B_{y_j} ist nicht 1–optimal.
Dann sei $i_0 := j$.

Fall b. Für jedes Kind y ist B_y 1–optimal.
Dann sei $i_0 \in \{1, \ldots, m\}$ beliebig.

- Ist $c(x) = 1$, so ist C_x undefiniert. Andernfalls ist

$$C_x := \{x_1\} \cup \bigcup_{i=1}^{m} \min\{B_{y_i}, D_{y_i}\}.^{10}$$

- Ist $c(x) = 1$, so ist D_x nicht definiert. Ebenso falls für ein Kind y die Menge B_y undefiniert ist (d.h. y ist ein Blatt in T). Andernfalls ist $D_x := \bigcup_{i=1}^{m} B_{y_i}$.

Nun kommen wir zu dem Fall, daß der aktuelle Baum nur noch aus genau einem Knoten w besteht. Wir unterscheiden wiederum zwei Fälle:

Fall 1. w ist ein 1–Knoten.
Ist $r(w) \neq \infty$, so nehmen wir w zu R hinzu.

Fall 2. w ist ein 2–Knoten.
Hier müssen wir zunächst überprüfen, ob die Knoten aus w bereits dominiert werden, d.h. ob $c(w) = 1$ ist.

Fall 2.1. $c(w) = 1$.
Die Knoten w_1 und w_2 werden bereits durch Knoten aus R dominiert. Wir nehmen daher $\min\{A_w, B_w, C_w, D_w\} = \min\{A_w, B_w\}$ zu R hinzu.

[10] Man beachte, daß eine der beiden Mengen B_{y_i}, D_{y_i} immer definiert ist!

Fall 2.2. $c(w) > 1$.

Die Knoten w_1 und w_2 werden nicht durch Knoten aus R dominiert. Daher müssen wir entweder A_w oder B_w zu R hinzunehmen. Falls B_w definiert und 1–optimal ist, nehmen wir B_w hinzu, andernfalls A_w.

R ist dann eine minimale r–dominierende Menge für G. Somit erhalten wir:

Satz 4.5.5
Das **RDS**–*Problem läßt sich auf* MExt^*(**Bäume**) *in Linearzeit lösen.*

Beweis. Die Korrektheit folgt unmittelbar induktiv aus den obigen Ausführungen. Somit brauchen wir nur noch bestätigen, daß sich der oben beschriebene Algorithmus in Linearzeit implementieren läßt. Eine starke Blätter Eliminationsordnung läßt sich nach Satz 4.5.3 in Linearzeit bestimmen. Problematisch ist nur die Aktualisierung der Mengenparameter. Denn würde man, wie im Algorithmus beschrieben, die Mengenparameter jedes 2–Knotens tatsächlich berechnen, so wäre die Laufzeit im schlechtesten Fall quadratisch in T (man beachte $O(\sum_{x \in V} |V(T_x)|) \subseteq O(|V(T)|^2)$). Um dieses Problem zu umgehen, speichern wir für einen Vater von Blättern x im aktuellen Baum, der ein 2–Knoten ist, folgendes ab:

Fall I. x besitzt nur Kinder, die 1–Knoten sind, und deren r–Wert ungleich 1 ist.
Dann speichern wir die tatsächlichen (definierten) Mengenparameter A_x, B_x, C_x und D_x ab.

Fall II. x besitzt einen 1–Knoten mit r–Wert 1 oder einen 2–Knoten als Kind.
Hier wird nur die Bildung der (definierten) Mengenparameter aus denen der Kinder abgespeichert, d.h. für jeden (definierten) Mengenparameter P_x von x wird für jedes Kind y abgespeichert, welcher Mengenparameter von y zu P_x (wie im Algorithmus beschrieben) hinzugefügt wird.

Zusätzlich wird in beiden Fällen noch der Optimalitätsgrad der definierten Mengenparameter aus $\{B_x, C_x, D_x\}$ und ein BOOLEsche Variable, die entscheidet, ob $D_x = \emptyset$ gilt, gemerkt.

Die abgespeicherten Informationen sind dann in $O(\mathrm{outdeg}(x))$. Im folgenden zeigen wir, daß sie auch in $O(\mathrm{outdeg}(x))$ berechnet werden können.

Zunächst berechnet man $|O|$ (O sei die Menge aus Fall 2, Akt. des A–Parameters, des obigen Algorithmus, siehe Seite 64). Danach werden die Informationen über A_x bestimmt (welcher Fall für A_x eintritt, läßt sich in $O(\mathrm{outdeg}(x))$ entscheiden). Dann wissen wir auch, wie die Kardinalität von A_x bzgl. der von O aussieht (s. Algorithmus, $|A_x| \in \{|O| + i : i = 0, 1, 2\}$). Anschließend werden die Informationen für B_x, C_x und D_x bestimmt. Hierbei läßt sich leicht feststellen, wie der Optimalitätsgrad der (definierten) B–, C– bzw. D–Mengenparameter aussieht (man benötigt ja nur die Kardinalität der Mengen B_x, C_x und D_x in Bezug auf $|O|$).

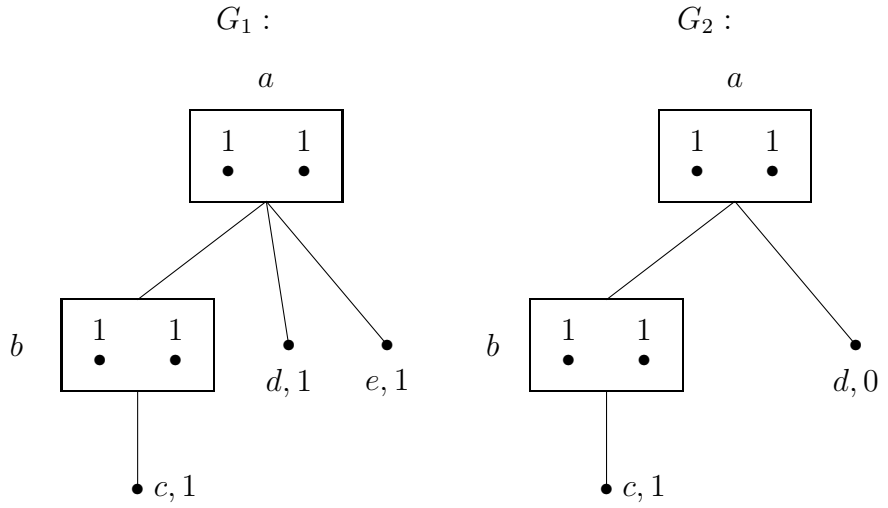
Zu zeigen bleibt, daß ein Mengenparameter, falls er zu R hinzugefügt wird (dies kann nur im Fall 1.2 ($x \neq w$) oder im Fall 2 ($x = w$) auftreten), effizient berechnet werden kann. Dazu sei x ein Knoten, für den ein Mengenparameter P_x benötigt wird. Wir gehen dann wie folgt (rekursiv) vor:[11]

Es sei $y \in V(T_x)$ ein 2–Knoten oder ein 1–Knoten, der im Fall 2 als ein 2–Knoten aufgefaßt wurde. Ferner sei P_y ein (definierter) Mengenparameter von y. Dann ist $\mathrm{Mengenparameter}(y, \text{'}P_y\text{'})$ die folgende Funktion:[12]

[11] Dieses Vorgehen läßt sich leicht mit Breitensuche implementieren.

[12] 'P_y' soll hierbei als Information aufgefaßt werden, daß der Mengenparameter P_y berechnet werden soll.

Abbildung 4.7: Zwei Beispiele zum **RDS**–Algorithmus auf MExt^*(**Bäume**)



Seien $z_1, \ldots, z_k$ die Kinder von y, denen Mengenparameter zugeordnet sind. Ist x ein 1–Knoten, der im Fall 2 als ein 2–Knoten aufgefaßt wurde, oder ist $k = 0$, so sei

$$\mathrm{Mengenparameter}(y, \text{'}P_y\text{'}) := P_y$$

(in diesem Fall wurde P_y bereits berechnet (s.o., Fall I)). Ist $k > 0$, so sei 'P_{z_i}', $i = 1, \ldots, k$, diejenige Information, die für das Kind z_i für den Parameter P_y abgespeichert wurde (s.o., Fall II). Dann setzen wir

$$\mathrm{Mengenparameter}(y, \text{'}P_y\text{'}) := I_y \cup \bigcup_{i=1}^{k} \mathrm{Mengenparameter}(z_i, \text{'}P_{z_i}\text{'}),$$

wobei

$$I_y := \begin{cases} \{y_1\}, & \text{falls '}P_y\text{'} \in \{\text{'}A_y\text{'}, \text{'}C_y\text{'}\}, \\ \emptyset, & \text{falls '}P_y\text{'} \in \{\text{'}B_y\text{'}, \text{'}D_y\text{'}\}. \end{cases}$$

Dann ist $P_x = \mathrm{Mengenparameter}(x, \text{'}P_x\text{'})$. Sei K die Menge der Knoten aus $V(T_x - x)$, denen ein Mengenparameter zugeordnet ist und dessen Vater kein 1–Knoten ist. Zur Berechnung von P_x wird dann die Zeit $O(t)$ benötigt, wobei t die Größe der Zusammenhangskomponente von $T_x(\{x\} \cup K)$ ist, in der x liegt.

Da ein Mengenparameter nur im Fall 1.2 ($x \neq w$) oder im Fall 2 ($x = w$) berechnet werden muß, benötigt man die Zeit $O(|V(T)|)$ zur Berechnung aller im Algorithmus benötigten Mengenparameter.

Insgesamt läßt sich der Algorithmus also in Linearzeit implementieren (man beachte $\sum_{v \in V} \mathrm{outdeg}(v) \leq \sum_{v \in V} \deg(v) = 2|E|$). ■

Wir betrachten nun noch zwei Beispiele zum obigen Algorithmus. In Abbildung 4.7 sind zwei knotenbewertete Graphen G_1 und G_2 aus MExt^*(**Bäume**) in Baumdarstellung dargestellt. Als starke Blätter Eliminationsordnung benützen wir $((b, \{c\}), (a, \{b, d, e\}), a)$ für G_1 und $((b, \{c\}), (a, \{b, d\}), a)$ für G_2. Nach dem Entfernen von c, bekommt b in beiden Graphen die Parameter $c(b) = \infty$, $A_b := \{b_1, c\}$, $B_b := \{c\}$, $C_b := \{b_1\}$, D_b ex. nicht zugeordnet.

In G_1 werden anschließend die Knoten d und e als 2–Knoten aufgefaßt, d.h. $c(d) = c(e) = \infty$, $A_d = \{d\}$, $A_e = \{e\}$, $D_d = D_e = \emptyset$ und B_d, C_d, B_e, C_e ex. nicht. Die Parameter für a werden dann (in G_1) wie folgt belegt: $c(a) = \infty$, $A_a := \{a_1\} \cup C_b \cup D_d \cup D_e = \{a_1, b_1\}$, $B_a := B_b \cup A_d \cup A_e = \{c, d, e\}$, $C_a := \{a_1, \} \cup B_b \cup D_d \cup D_e = \{a_1, c\}$, D_a ex. nicht. Damit ist $R \cup A_a = \{a_1, b_1\}$ eine minimale r–dominierende Menge in G_1.

In G_2 wird nach dem Entfernen von c zunächst auch d entfernt und $R := \{d\}$, $c(a) = 1$ gesetzt. Die Parameter für a werden dann wie folgt belegt: $c(a) = 1$, $A_a := \{a_1\} \cup C_b = \{a_1, b_1\}$ (oder $A_a := \{a_1\} \cup B_b = \{a_1, c\}$), $B_a := B_b = \{c\}$, C_a und D_a ex. nicht. Damit ist $R \cup B_a = \{c, d\}$ eine minimale r–dominierende Menge in G_2.

Diese beiden Beispiele zeigen einen wichtigen Aspekt: Bei den Algorithmen für das **RDS**–Problem auf Bäumen ([Sla76], [HY90], [BCD94]) kann nach dem Entfernen eines Blattes entschieden werden, ob es zur minimalen r–dominierenden Menge hinzugefügt werden darf bzw. muß, oder nicht. Hier ist ein solches Vorgehen nicht möglich, denn: Man sieht leicht, daß es keine minimale r–dominierende Menge für G_1 gibt, die c enthält, und daß $\{c, d\}$ die einzige minimale r–dominierende Menge für G_2 ist. Somit kann nach dem Entfernen von c noch nicht entschieden werden, ob c zur Menge R hinzugefügt werden muß oder nicht.

4.6 $\mathbb{NP}$–vollständige Probleme auf modularen Erweiterungen

Es seien $G = (V, E)$ ein Graph und $x \notin V$. Dann sei $G \bowtie x := (V', E')$ derjenige Graph, den man erhält, wenn man x mit jedem Knoten in G verbindet, d.h.

$$V' := V \cup \{x\}, \qquad E' := E \cup \{vx : v \in V\}.$$

Es sei $G = (V, E)$ ein Graph. Folgende Graphenparameter sind grundlegend für die algorithmische Graphentheorie:

$\alpha(G) := \max\{|I| : I \subseteq V, G(I)$ unabhängig$\}$,

$\omega(G) := \alpha(\overline{G}) = \max\{|C| : C \subseteq V, G(C)$ vollständig$\}$,

$\chi(G) := \min\{k : V_1, \ldots, V_k$ eine Partition von $V, G(V_1), \ldots, G(V_k)$ unabhängig$\}$,

$\kappa(G) := \chi(\overline{G}) = \min\{k : V_1, \ldots, V_k$ eine Partition von $V, G(V_1), \ldots, G(V_k)$ vollständig$\}$.

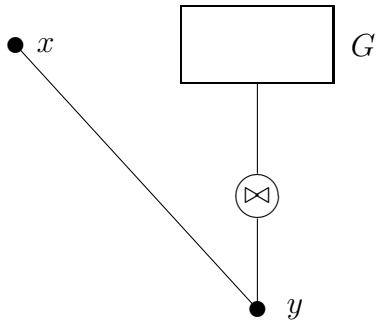
$\chi(G)$ nennt man auch die *chromatische Zahl* von G, da $\chi(G)$ die minimale Anzahl von Farben angibt, die man benötigt, um die Knoten von G so zu färben, daß je zwei benachbarte Knoten nicht die gleiche Farbe zugewiesen bekommen.

Insbesondere für perfekte Graphen spielen obige Graphenparameter eine entscheidene Rolle. G heißt *α–perfekt* (bzw. *χ–perfekt*), falls für jedes $V' \subseteq V$ gilt $\alpha(G(V')) = \kappa(G(V'))$ (bzw. falls für jedes $V' \subseteq V$ gilt $\chi(G(V')) = \omega(G(V'))$). G heißt perfekt, falls G α–perfekt oder χ–perfekt ist. Dann gilt der folgende

Satz 4.6.1 (Perfect Graph Theorem, [Lov72])
Für einen Graphen G sind folgende Aussagen äquivalent:

(1) G ist perfekt.

Abbildung 4.8: Der Graph $G' := (G + x) \bowtie y$



(2) G *ist* α*–perfekt.*

(3) G *ist* χ*–perfekt.* ■

Besonders im Hinblick auf algorithmischen Nutzen spielen die perfekten Graphen eine zentrale Rolle (vgl. [Gol80]).

Modulare Erweiterungen einer Graphenklasse sind dagegen meist nicht perfekt, denn: Ein C_5 ist nicht perfekt, da $\chi(C_5) = 3 \neq 2 = \omega(G)$. Damit ist auch der Graph $C_5 \bowtie x$ nicht perfekt, aber z.B. $C_5 \bowtie x \in \mathrm{MExt}^*(\{K_2\})$.

Satz 4.6.2
Auf der Graphenklasse $\mathbf{G} := \{G \bowtie x : G \text{ Graph}, x \notin V(G)\}$ *sind folgende Probleme* $\mathbb{NP}$*–vollständig:*

(1) Die Berechnung der Graphenparameter $\alpha(G)$, $\omega(G)$, $\kappa(G)$ *und* $\chi(G)$.

(2) Das **HC***–Problem (*HAMILTON*kreis Problem).*

(3) Das **HP***–Problem (*HAMILTON*pfad Problem).*

(4) Das **RDHC***– und* **RDHP***–Problem.*

Beweis. Wir benutzen Satz 1.2.2, denn die Berechnung der Graphenparameter, das **HC**– und das **HP**–Problem sind für beliebige Graphen $\mathbb{NP}$–vollständig ([GJ79], [Bra94]).

Es sei G ein Graph, $x \notin V(G)$ und $G' := G \bowtie x$. Dann gilt

$$\alpha(G') := \alpha(G), \quad \omega(G') := \omega(G) + 1, \quad \kappa(G') := \kappa(G), \quad \chi(G') := \chi(G) + 1,$$

woraus (1) folgt.

(2) sieht man so: G besitzt genau dann einen HAMILTONpfad, wenn G' einen HAMILTONkreis besitzt. (Denn ist P ein HAMILTONpfad in G, so ist $x - P - x$ ein HAMILTONkreis in G'. Weiter ist jeder HAMILTONkreis in G' von der Form $x - P - x$, P ein HAMILTONpfad in G.)

Seien G ein Graph und $x, y \notin V(G)$. Weiter sei $G' := (G + x) \bowtie y$ (vgl. Abbildung 4.8). Dann gilt

$$G \text{ hat einen HAMILTONpfad} \iff G' \text{ hat einen HAMILTONpfad}.$$

Ist P ein HAMILTONpfad in G, so ist $P - y - x$ ein HAMILTONpfad in G'. Umgekehrt beginnt (oder endet) jeder HAMILTONpfad P' in G' bei x, P' hat also die Form $P' = x - y - P$ (oder $P' = P - y - x$), P ein HAMILTONpfad in G.

(4) folgt unmittelbar daraus, daß ein Graph G genau dann einen HAMILTONpfad (bzw. HAMILTONkreis) besitzt, falls der knotenbewertete Graph (G, r), $r : V(G) \to \mathbb{N}$, $r \equiv 0$, einen r–dominierenden Pfad (bzw. Kreis) besitzt. ■

Da $\mathrm{MExt}^*(\textbf{Bäume}) \supset \{G \bowtie x : G \text{ Graph}, x \notin V(G)\}$ erhalten wir sofort:

Korollar 4.6.3
Auf der Graphenklasse $\mathrm{MExt}^*(\textbf{Bäume})$ *sind folgende Probleme* $\mathbb{NP}$*–vollständig:*

(1) Die Berechnung der Graphenparameter $\alpha(G)$, $\omega(G)$, $\kappa(G)$ *und* $\chi(G)$.

(2) Das **HC***–Problem.*

(3) Das **HP***–Problem.*

(4) Das **RDHC***– und* **RDHP***–Problem.* ■

Literaturverzeichnis

[ADKP89] K. ABRAHAMSON, N. DADOUN, D. G. KIRKPATRICK, AND T. PRZYTYCKA. A simple parallel tree contraction algorithm. *Journal of Algorithms 10*, pages 287–302, 1989.

[AHU74] A. AHO, J. HOPCROFT, AND J. ULLMAN. The design and analysis of computer algorithms. *Addison–Wesley*, 1974.

[BCD94] A. BRANDSTÄDT, V. D. CHEPOI, AND F. F. DRAGAN. The algorithmic use of hypertree structure and maximum neighbourhood orderings. *Technical Report SM-DU-244, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1994. 20th International Workshop "Graph–Theoretic Concepts in Computer Science" 1994, Springer, Lecture Notes in Computer Science 903, pages 65–80. Theoretical Computer Science (to appear).*

[BD94] A. BRANDSTÄDT AND F. F. DRAGAN. A linear time algorithm for connected r–domination and Steiner Tree on Distance–Hereditary Graphs. *Technical Report SM-DU-261, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1994.*

[BDCV93] A. BRANDSTÄDT, F. F. DRAGAN, V. D. CHEPOI, AND V. I. VOLOSHIN. Dually chordal graphs. *Technical Report SM-DU-225, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1993. 19th International Workshop "Graph–Theoretic Concepts in Computer Science" 1993, Springer, Lecture Notes in Computer Science 790 (Jan van Leeuwen, ed.), pages 237–251.*

[BDN94] A. BRANDSTÄDT, F. F. DRAGAN, AND F. NICOLAI. Homogeneously orderable graphs. *Technical Report SM-DU-271, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1994. 21th International Workshop "Graph–Theoretic Concepts in Computer Science" 1995, Springer, Lecture Notes in Computer Science (to appear).*

[BK87] A. BRANDSTÄDT AND D. KRATSCH. Domination problems on permutation and other graphs. *Theoretical Computer Science 54*, pages 181–198, 1987.

[Bra93] A. BRANDSTÄDT. Special graph classes — a survey (revised version). *Technical Report SM-DU-199, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1993.*

[Bra94] A. BRANDSTÄDT. Graphen und Algorithmen. *B. G. Teubner, Stuttgart*, 1994.

[Bun74] A. BUNEMAN. A characterization of rigid circuit graphs. *D.M. 9*, pages 205–212, 1974.

[Col86] R. COLE. Parallel merge sort. *FOCS*, 1986.

[CP84] D. G. CORNEIL AND Y. PERL. Clustering and domination in perfect graphs. *D.A.M. 9*, pages 27–39, 1984.

[CV86a] R. COLE AND U. VISHKIN. Approximate and exact parallel scheduling with applications to list, tree and graph problems. *FOCS*, 1986.

[CV86b] R. COLE AND U. VISHKIN. Deterministic coin tossing and acceleration cascades: Micro and macro techniques for designing parallel algorithms. *Proceesings of the 18th Annual ACM Symposium on Theory of Computing*, pages 206–219, 1986.

[Dah95] E. DAHLHAUS. Efficient parallel modular decomposition. *21th International Workshop "Graph–Theoretic Concepts in Computer Science", Springer, Lecture Notes in Computer Science (to appear)*, 1995.

[DB94] F. F. DRAGAN AND A. BRANDSTÄDT. r–dominating cliques in Helly graphs and chordal graphs. *Proc. of the 11th STACS, Caen, France, Springer, LNCS 775*, 1994.

[DMS88] A. D'ATRI, M. MOSCARINI, AND A. SASSANO. The steiner tree problem and homogeneous sets. *MFCS '88, Springer, Lecture Notes in Computer Science 324*, pages 246–261, 1988.

[DN95] F. F. DRAGAN AND F. NICOLAI. r-domination problems on homogeneously orderable graphs. *Technical Report SM-DU-275, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1995. Accepted for FCT 1995.*

[Dra93] F. F. DRAGAN. HT–graphs: centers, connected r–domination and Steiner Trees. *Computer Science Journal of Moldova Vol. 1, No. 2*, pages 64–83, 1993.

[Dra94] F. F. DRAGAN. Dominating cliques in distance–hereditary graphs. *"Algorithm Theory – SWAT '94" 4th Scandinavian Workshop on Algorithm Theory Aarhus, Denmark, July 1994, Springer, LNCS 824 (Erik M. Schmidt and Sven Skyum, eds.)*, 1994.

[Duc76] P. DUCHET. Propriete de Helly et problemes de representation. *In Colloqu. Internat. CNRS 260, Problemes Combinatoires et Theorie du Graphs, Orsay, France*, pages 117–118, 1976.

[Fla78] C. FLAMENT. Hypergraphs arbores. *D.M. 21*, pages 223–226, 1978.

[Gav74] F. GAVRIL. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Comb. Th. (B) 16*, pages 47–56, 1974.

[GJ79] M. R. GAREY AND D. S. JOHNSON. Computers and Intractability: A guide to the theory of NP–completeness. *W. H. Freeman*, 1979.

[Gol80] M. C. GOLUMBIC. Algorithmic graph theory and perfect graphs. *Academic Press*, 1980.

[GR88] A. GIBBONS AND W. RYTTER. Efficient parallel algorithms. *Cambridge University Press*, 1988.

[Har94] T. J. HARRIS. A survey of PRAM simulation techniques. *ACM Comp. Surveys, Vol. 26, No. 2*, pages 185–206, 1994.

[HHM89] R. B. HAYWARD, C. HOANG, AND F. MAFFRAY. Optimizing weakly triangulated graphs. *Graphs and Combin. 5*, pages 339–349, 1989.

[HM90] P. L. HAMMER AND F. MAFFRAY. Completely separable graphs. *D.A.M. 27*, pages 85–99, 1990.

[HY88] X. HE AND Y. YESHA. Binary tree algebraic computation and parallel algorithms for simple graphs. *Journal of Algorithms 9*, pages 92–113, 1988.

[HY90] Y. HE AND Y. YESHA. Efficient parallel algorithms for r-dominating set and p-center problems on trees. *Algorithmica*, pages 129–145, 1990.

[Ja92] J. JA JA. An Introduction to parallel algorithm. *Addison–Wesley*, 1992.

[KDL94] D. KRATSCH, P. DAMASCHKE, AND A. LUBIW. Dominating cliques in chordal graphs. *D.M. 128*, pages 269–275, 1994.

[Lov72] L. LOVASZ. Normal hypergraphs and the perfect graph conjecture. *D.M. 2*, pages 253–267, 1972.

[Lub87] A. LUBIW. Doubly lexical orderings of matrices. *SIAM J. Comput. 16, No. 5*, pages 854–879, 1987.

[MB87] H. MÜLLER AND A. BRANDSTÄDT. The NP–completeness of Steiner Tree and Dominating Set for chordal bipartite graphs. *Theor. Comp. Science 53*, pages 257–265, 1987.

[MS94] R. MCCONNEL AND J. SPRINRAD. Linear–time modular decomposition and efficient transitive orientation of comparability graphs. *In 5th Annual ACM–SIAM Symposium of Discrete Algorithms (SODA)*, pages 536–545, 1994.

[Nic94a] F. NICOLAI. A hypertree characterisation of distance–hereditary graphs. *Technical Report SM-DU-255, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1994.*

[Nic94b] F. NICOLAI. Strukturelle und algorithmische Aspekte distanz–erblicher Graphen und verwandter Klassen. *Dissertation, Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1994.*

[NS95] F. NICOLAI AND T. SZYMCZAK. Homogeneous extensions of trees and r–domination problems. *Technical Report (to appear), Gerhard–Mercator–Universität — Gesamthochschule Duisburg, 1995.*

[Ola89] S. OLARIU. Weak bipolarizable graphs. *D.M. 74*, pages 159–171, 1989.

[PT86] R. PAIGE AND R.E. TARJAN. Three Partition Refinement Algorithms. *SIAM J. Comp. Vol. 16 No. 5*, 1986.

[Rei90] K. R. REISCHUK. Einführung in die Komplexitätstheorie. *B. G. Teuber, Stuttgart*, 1990.

[RTL76] D. ROSE, R. E. TARJAN, AND G. LUEKER. Algorithmic aspects on vertex elimination on graphs. *SIAM J. Comput. Vol. 5*, pages 266–283, 1976.

[Sei74] D. SEINSCHE. On a property of the class of n-colorable graphs. *J. Comb. Th. (B) 16*, pages 191–193, 1974.

[Sla76] P. L. SLATER. R–domination in graphs. *J. Assoc. Comput. Mach., 23 (3)*, pages 446–450, 1976.

[Spi93] J.P. SPINRAD. Doubly Lexical Ordering of Dense 0-1 Matrices. *I.P.L. 45*, pages 229–235, 1993.

[SV82] Y. SHILOACH AND U. VISHKIN. An $O(\log n)$ parallel connectivity algorithm. *Journal of Algorithms 3*, pages 57–67, 1982.

[SV88] B. SCHIEBER AND U. VISHKIN. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput. Vol. 17, No. 6*, pages 1253–1262, 1988.

[TY84] R. E. TARJAN AND M. YANNAKAKIS. Simple linear–time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput. Vol. 13, No. 3*, pages 566–579, 1984.

[Vis83] U. VISHKIN. Synchronous parallel computation — A survey. *Technical Report TR-71, Department of Computer Science, Courant Institute, New York University, 1983.*

[Wal72] J.R. WALTER. Representations of rigid cycle graphs. *PhD thesis, Wayne State Univ.*, 1972.

Symbol– und Sachwortverzeichnis

Erklärung

Hiermit bestätige ich, daß ich die vorliegende Arbeit selbständig verfaßt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

(Datum) (Unterschrift)