Diplomarbeit

Algorithmische Aspekte und Komplexität von Power Domination in Netzwerken

Daniel Raible

Betreuer: Prof. Dr. Klaus-Jörn Lange
Prof. Dr. Rolf Niedermeier
Dipl.-Inform. Jiong Guo

begonnen am: 02.01.2005
beendet am: 30.07.2005

Arbeitsbereich Theoretische Informatik/Formale Sprachen
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Zusammenfassung

Das Power Dominating Set Problem ist eine Variante des wohl bekannten Domination-Problems in Graphen: Finde für einen ungerichteten Graphen $G(V, E)$ eine kleinstmögliche Teilmenge $P \subseteq V$, so dass alle Knoten observiert sind. Diesbezüglich observiert ein Knoten aus P sich selbst und alle seine Nachbarn, und falls ein observierter Knoten, bis auf einen, nur observierte Nachbarn besitzt, so folgt daraus, dass der unobservierte Knoten ebenso observiert wird. In dieser Arbeit zeigen wir unter anderem die **NP**-Vollständigkeit des Power Dominating Set Problems, und dass es **NP**-vollständig bleibt, falls wir es auf planare, Circle- und Split-Graphen beschränken. Die aufgeführte Reduktion lässt darüber hinaus folgern, dass das Power Dominating Set Problem sich nicht besser approximieren lässt als das Dominating Set Problem und wie dieses $W[2]$-hart ist. Weitere Ergebnisse sind Linearzeit-Algorithmen für das Power Dominating Set Problem auf Bäumen und Generalisierten Series-Parallel Graphen, sowie ein **FPT**-Algorithmus für allgemeine Graphen mit Baumweite als Parameter. Der Algorithmus für Bäume vereinfacht den schon bestehenden von Haynes et al. [10] erheblich. Für die beiden weiteren Graphklassen finden wir eine äquivalente, aber aussagekräftigere Formulierung des Problems und lösen dieses mit Hilfe von Dynamischem Programmieren auf der Baumzerlegung.

Ich versichere hiermit, dieses Dokument selbständig angefertigt und nicht mehr als die zitierten Quellen hinzugezogen zu haben.

Daniel Raible

Danksagungen

Jiong Guo, Rolf Niedermeier, Jens Gramm, Falk Hüffner, Markus Geyer, Stefanie Reifferscheid, Bianca Kuon, Maria Theresa Binkele, Jennifer Portillo, Anja Korsten, Robert Schweickert

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Problemdefinition

Das Power Dominating Set Problem wird durch die Notwendigkeit zur Überwachung elektrischer Netzwerke motiviert. Man möchte das Netzwerk bezüglich gewisser elektrischer Größen überwachen. Dazu platziert man an speziellen Stellen im Netzwerk Messstationen, so genannte PMU's (phase measurement unit). Man möchte nun die Anzahl der PMU's minimieren. Dies lässt sich nun auch in der Sprache der Graphen ausdrücken. Ein Graph $G(V, E)$ stellt das elektrische Netzwerk dar. Eine PMU kann auf einem Knoten platziert werden. Dieser Knoten wird dann als *observierend* bezeichnet. Ziel ist es, dass alle Knoten und Kanten *observiert* sind. Die Regeln, nach denen sich die jeweilige *Observation* für Knoten und Kanten bestimmen lässt, sind die folgenden:

1. Ist eine PMU auf einem Knoten v platziert, so sind v und alle seine inzidenten Kanten observiert.
2. Jeder Knoten der zu einer observierten Kante inzident ist, ist observiert.
3. Jede Kante die zwei observierte Knoten verbindet, ist observiert.
4. Ist ein Knoten v zu insgesamt $k > 1$ Kanten inzident und sind $k-1$ bereits observiert, so folgt, dass auch die übrige k-te Kante observiert ist.

Eine *Power Dominating Set* (pds) ist nun eine Menge $P \subseteq V$, so dass alle Knoten $v \in P$ observierend sind und der gesamte Graph G observiert ist.
Das Power Dominating Set Problem lässt sich dann so formulieren:

Power Dominating Set (PDS)
Eingabe: Ein Graph $G(V, E)$.
Ziel: Eine kleinstmögliche Menge $P \subseteq V$, die eine pds für G ist.

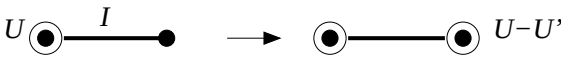
1.2 Power Dominating Set und elektrische Netzwerke

Wie bereits erwähnt gibt es eine enge Verbindung zwischen Power Dominating Set und elektrischen Netzwerken. Diese Verbindung besteht darin, dass

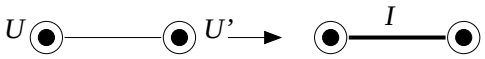
man für jede der Regeln ein korrespondierendes Gesetz aus der Elektrotechnik findet, welches die Regel begründet. Wir zeigen jetzt, wie man diese Regeln aus dem Ohmschen Gesetz und der Kirchhoffschen Regel gewinnt. Im Weiteren bedeutet eine dicke Linie eine observierte Kante und eine dünne Linie eine unobservierte Kante. Gleichermaßen besitzen observierte Knoten eine Umrandung und nicht observierte keine.

1.Regel Ist eine PMU auf einem Knoten platziert, so kennt sie die am Knoten herrschende Spannung und den jeweiligen Strom in den inzidenten Kanten.

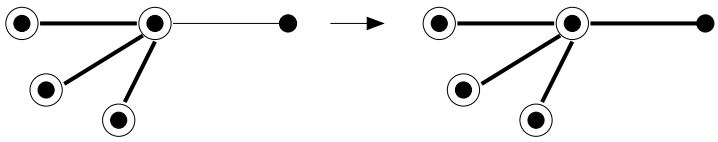
2.Regel Wir kennen die Spannung U am observierten Knoten, den Widerstand R der Leitung und den Strom I, der durch die Leitung fließt. $U' = I \cdot R$ ist dann der Spannungsabfall. Somit herrscht am unobservierten Knoten die Spannung $U - U'$.



3.Regel Wir kennen die Spannungen U, U' der beiden observierten Knoten und den Wiederstand der Leitung. Somit ist $I = \frac{U-U'}{R}$ der Strom, der durch die Leitung fließt.



4.Regel Wir kennen den Strom in $k-1$ Leitungen die am Knoten anliegen und den Gesamtstrom I. Ist I' der Gesamtstrom der $k-1$ Leitungen, so fließt der Strom $I - I'$ durch die k-te Leitung.



1.3 Definitionen und Notation

Alle Graphen, die betrachtet werden, werden als einfach, ungerichtet und ohne Schleifen angenommen, falls nichts Gegenteiliges erwähnt wird. Wir nennen einen Graphen $G'(V', E')$ einen *induzierten Subgraphen* eines Graphen $G = (V, E)$ genau dann, wenn gilt $V' \subseteq V$ und $E' = \{\{u, v\} \mid u, v \in V' \text{ und } \{u, v\} \in E\}$.
Für einen Knoten $v \in V$ in einem Graphen $G(V, E)$ ist $N_G(v)$ die Menge seiner Nachbarn in G, also $N_G(v) = \{u \mid \{u, v\} \in E\}$. Wir setzen noch $N_G[v] := N_G(v) \cup \{v\}$ und für eine Menge $V' \subseteq V$ analog $N_G[V'] := \cup_{v \in V'} N[v]$. $N_G^E(v)$ ist die Menge der zu v inzidenten Kanten in E, also $N_G^E(v) = \{\{u, v\} \mid \{u, v\} \in E\}$. Der *Grad* eines Knotens ist die Anzahl seiner Nachbarn. Auf den Grad eines Knotens v beziehen wir uns mit $\delta_G(v)$. Es gilt $\delta_G(v) = |N_G(v)|$. Die Indizierung mit G wird unterlassen, falls sie sich aus dem Kontext erschließt. Auf den größten Grad eines Knotens in G beziehen wir uns mit $\Delta(G)$, also $\Delta(G) = \max\{\delta(v) \mid v \in V\}$. Ist V' eine Knotenmenge in einem Graphen

$G(V, E)$, so ist $E(V') = \{\{u, v\} \mid u, v \in V', \{u, v\} \in E\}$. Eine Knotenmenge $\{v_1, \ldots, v_k\}$ ist ein Pfad in $G(V, E)$, falls $\{v_i, v_{i+1}\} \in E$ für $1 \leq i < k$ gilt. Mit $P = q_1, \ldots, q_k$ bezeichnen wir einen ungerichteten Pfad $\{q_1, \ldots q_k\}$ zwischen q_1 und q_k. Mit $P = (q_1, \ldots, q_k)$ wird ein gerichteter Pfad zwischen q_1 und q_k bezeichnet. Betrachten wir einen gerichteten Graphen $G(V, E)$ so legen wir für eine Teilmenge $F \subseteq E$ dann $indeg_F(v) := |\{(u, v) \mid (u, v) \in F\}|$ und $outdeg_F(v) := |\{(v, u) \mid (v, u) \in F\}|$ fest.

1.4 Parametrisierte Komplexitätstheorie

In dieser Arbeit brauchen wir an diversen Stellen Konzepte aus der parametrisierten Komplexitätstheorie. In diesem kurzen Abschnitt führen wir deswegen auf informellem Wege in die benötigten Konzepte ein. Parametrisierte Komplexität ist eine Klassifizierungsstruktur, die von zwei Größen abhängt, um die Komplexität von Problemen zu messen. Die erste ist wie in der klassischen Komplexitätstheorie die Problemgröße n. Die zweite Größe ist ein *Parameter* k, der zumeist eine positive ganze Zahl ist. Ein Problem wird *fixed-parameter tractable* genannt, falls es in $f(k) \cdot n^{O(1)}$ Zeit gelöst werden kann, wobei f eine berechenbare Funktion ist, die nur von k abhängt. Eine Theorie um *fixed-parameter intractability* nachzuweisen wurde von Downey und Fellows [17] entwickelt, die hierzu das Konzept der *parametrisierten Reduktion* einführten. Eine parametrisierte Reduktion ist eine Funktion, die eine parametrisierte Sprache L auf eine weitere parametrisierte Sprache L' reduziert. Sie berechnet in Zeit $f(k) \cdot n^{O(1)}$ für eine Instanz $(x, k) \in L$ eine Instanz $(x', k') \in L'$, wobei k' nur von k abhängt und $(x, k) \in L \iff (x', k') \in L'$ gilt. Die grundlegende Komplexitätsklasse um fixed-parameter intractability zu zeigen, ist $W[1]$. Es gibt einige schwerwiegende Gründe anzunehmen, dass ein $W[1]$-hartes Problem fixed-parameter intractable ist [17]. Basierend auf $W[1]$ führen Downey und Fellows zudem noch eine Hierarchie von Komplexitätsklassen $W[i]$ mit $i \geq 1$ ein, sodass $W[i] \subseteq W[i+1]$ gilt.

1.5 Vereinfachung der Regeln

Die in Abschnitt 1.1 angegebenen Regeln müssen auf Knoten und Kanten angewendet werden. Deswegen haben sie den Anschein schwer handhabbar zu sein. Ebenfalls sind vier Regeln recht umfangreich. Diese Regeln sind durch die direkte Übertragung aus den entsprechenden elektrophysikalischen Gegebenheiten entstanden. Es ist nun möglich, diese vier Regeln durch zwei zu ersetzen, welche sich nur auf die Knoten des Graphen beziehen. Sei nun ein Graph $G(V, E)$ und $P \subseteq V$ gegeben. Dann sind die vereinfachten Regeln:

- $\mathcal{L}$: Alle $v \in N[p]$ mit $p \in P$ sind observiert (Lokale Regel).
- $\mathcal{G}$: Existiert ein observierter Knoten v mit $\delta(v) = k > 1$ und sind bereits $k-1$ Nachbarn von v observiert, so folgt, dass auch der k-te Nachbar observiert ist (Globale Regel).

Wir werden nun zeigen, dass die neu eingeführten Regeln äquivalent zu den alten sind. Das heißt, dass eine Menge $P \subseteq V$ alle Knoten genau dann mit den alten Regeln observiert, wenn sie auch alle Knoten mit den neuen observiert. Dies

wird bereits in J. Kneis et al [15] gezeigt. Hier stellen wir nun einen alternativen und ausführlicheren Beweis dafür vor:

Proposition 1.1. *Sei $G(V, E)$ ein Graph und seien OV_a die Knoten, die mit den Regeln 1-4 aus Abschnitt 1.1 observiert werden und OV_n diejenigen Knoten, die mit den Regeln $\mathfrak{L}$ und $\mathcal{G}$ observiert werden. Dann gilt $OV_a = OV_n$.*

Beweis.

$OV_n \subseteq OV_a$: Wir können die Regel $\mathfrak{L}$ simulieren indem wir zuerst Regel 1 und sogleich Regel 2 ausführen. Sei V_0 die Menge der Knoten, die wir durch maximale Anwendung der Regel $\mathfrak{L}$ erhalten und V_ℓ, $\ell > 0$, die Menge der Knoten, die nach der ℓ-ten Anwendung von Regel $\mathcal{G}$ observiert sind. Sei $v \in V_0$ ein Knoten auf den wir Regel $\mathcal{G}$ anwenden können und $N(v) = \{v_1, \ldots, v_k\}$ seine Nachbarschaft. Da o.B.d.A $N[v] \setminus \{v_k\} \subseteq V_0$ gilt, folgt dass alle Kanten $\{v, v_i\}$, $1 \leq i < k$, bezüglich der Regeln 1-4 observiert sind und $V_1 = V_0 \cup \{v_k\}$ gilt. Wenn wir dann Regel 4 auf v und danach Regel 2 anwenden, so sind auch $\{v, v_k\}$ und v_k bezüglich der alten Regeln observiert. Somit können nun auch alle Knoten, die in V_1 sind, mit den Regeln 1-4 observiert werden. Indem wir auf diese Art sukzessive die Anwendung der Regel $\mathcal{G}$ nachbilden, folgt, dass wir die Menge der Knoten V_s, die aus der letzten Anwendung von $\mathcal{G}$ resultiert, ebenfalls mit den Regeln 1-4 observieren können. Da $V_s = OV_n$ und $V_s \subseteq OV_a$, folgt $OV_n \subseteq OV_a$.

$OV_a \subseteq OV_n$: Sei V_0' die Menge der Knoten, die durch maximale Anwendung der Regel 1 und 2 entsteht. Dann ist klar, dass $V_0' = V_0$ gilt. Sei $v \in V_0'$ ein Knoten auf den wir Regel 4 anwenden können. Indem wir danach sofort Regel 2 anwenden wird ein weiterer Knoten v' observiert. Falls möglich wenden wir danach Regel 3 an. Hierdurch entsteht nun die Knotenmenge V_1', in der alle Knoten und alle Kanten zwischen diesen Knoten bezüglich der Regeln 1-4 observiert sind. Klar ist aber auch, dass wir v' durch Anwendung von $\mathcal{G}$ auf v bezüglich der neuen Regeln observieren können. Führen wir dies fort bis keine Anwendung der Regel 4 mehr möglich ist, so erhalten wir Knotenmengen V_s' und V_s mit $V_s = V_s' = OV_a$, die auch mittels der neuen Regeln observiert werden. Da $V_s \subseteq OV_n$, folgt $OV_a \subseteq OV_n$

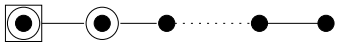
□

Mit den Regeln $\mathfrak{L}$ und $\mathcal{G}$ können wir demnach genau dieselben Knoten observieren wie mit den Regeln 1-4. Dann sind aber auch alle Kanten observiert und die beiden Regelsätze äquivalent.
Aufgrund größerer Einfachheit werden wir im Rest dieser Arbeit nur noch die Regeln $\mathfrak{L}$ und $\mathcal{G}$ verwenden und die Regeln 1-4 nicht mehr betrachten.

1.6 Beispiele zur Anwendung der Observationsregeln

Es kommt nun darauf an, mit Hilfe des Zusammenspiels dieser Regeln einen zusammenhängenden Graphen zu observieren. Deshalb wollen wir anhand der folgenden Beispiele etwas Routine im Anwenden der Regeln vermitteln. Betrachten wir nun zwei Graphen und ihre kleinstmögliche pds.

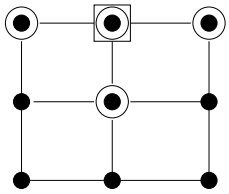
Beispiel 1.2. Als erstes betrachten wir einen einfachen Pfad. Die obervierten Knoten sind zusätzlich umrandet und die observierenden werden von einem Quadrat umschlossen.



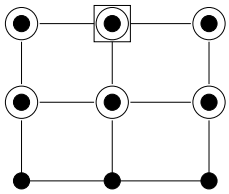
Mit Regel $\mathcal{L}$ sind sofort der 1. und 2. Knoten observiert.

Die weiteren Knoten werden durch eine sukzessive Anwendung der Regel $\mathcal{G}$ observiert. Ist u der Knoten der im Moment am weitesten rechts und observiert ist, so besitzt er immer exakt einen weiteren observierten Nachbarn und einen unobservierten Nachbarn v. Wird $\mathcal{G}$ auf u angewendet, so ist v observiert. Damit ist dann v der Knoten, der am weitesten rechts steht und observiert ist.

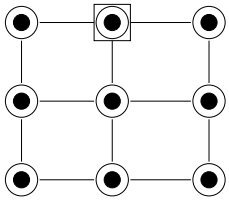
Beispiel 1.3. Das nächste Beispiel ist ein 3×3-Grid-Graph:



Der observierende Knoten und seine Nachbarschaft sind sofort durch Regel $\mathcal{L}$ observiert.



Der linke und rechte Nachbar des observierenden Knotens, haben die Voraussetzung zur Anwendung von Regel $\mathcal{G}$ erfüllt.



Alle unobservierten Knoten konnten, dadurch observiert werden, dass auf den jeweils observierten Nachbarknoten Regel $\mathcal{G}$ angewendet wurde.

1.7 Bisherige Resultate

Die bisher gewonnenen Resultate über PDS sind momentan noch nicht weitreichend und die meisten sind in T.W. Haynes, S.M. Hedetniemi, S.T. Hedetniemi

und M.A. Henning [10] zu finden. Ein wichtiges Ergebnis ist der dort geführte **NP**-Vollständigkeitsbeweis. Es wird gezeigt, dass die folgende Entscheidungsproblemvariante bereits für bipartite und chordale Graphen NP-vollständig ist:

Eingabe: Ein Graph $G(V, E)$.
Ziel: Gibt es für G eine pds der Größe k?

Eine einfache Modifikation des Standardbeweises für die **NP**-Vollständigkeit von Dominating Set in T.W. Haynes, S.T. Hedetniemi und P.J. Slater [11] ist die Grundlage, um dasselbige für Power Dominating Set zu zeigen; sogar dann wenn die gegebene Graphinstanz auf bipartite oder chordale Graphen beschränkt ist. Es wurde hierfür 3-SAT auf Power Dominating Set reduziert. Wir können also im Allgemeinen nicht damit rechnen für beliebige Graphinstanzen einen polynomiellen Algorithmus zur Lösung von Power Dominating Set zu finden.
M. Dorfling und A. Henning [18] untersuchten die spezielle Klasse der $n \times m$-Grid-Graphen. Das hauptsächliche Ergebnis dieser Arbeit betrifft die minimale Größe einer Power Dominating Set $\gamma_P(G)$ eines Grid-Graphen. Ist G ein $n \times m$-Grid-Graph, $m \geq n \geq 1$, so zeigen sie:

$$\gamma_P(G) = \begin{cases} \lceil \frac{n+1}{4} \rceil : n \equiv 4 (\text{mod } 8). \\ \lceil \frac{n}{4} \rceil : \textit{sonst}. \end{cases}$$

Die Ähnlichkeit der **NP**-Vollständigkeitsbeweise von Dominating Set und Power Dominating Set scheinen nicht nur zufälligerweise zu existieren. Dies ist bereits augenscheinlich, wenn man sich die Problemdefinition anschaut.

Dominating Set (DS)
Eingabe: Ein Graph $G(V, E)$.
Ziel: Eine kleinstmögliche Menge $D \subseteq V$, so dass für alle $v \in V$ entweder $v \in D$ gilt oder es ein $s \in D$ mit $v \in N(s)$ gibt.

Dominating Set ist nichts anderes als Power Dominating Set auf Regel $\mathfrak{L}$ eingeschränkt. Eine einfache Beobachtung ist, dass jede Dominating Set D auch eine Power Dominating Set ist. Da es für alle Knoten $v \in V$ ein $d \in D$ mit $v \in N[d]$ gibt, folgt dass alle Knoten mit Regel $\mathfrak{L}$ observiert sind. Sei nun $\gamma(G)$ die Größe einer kleinstmöglichen Dominating Set für den Graphen G, dann gilt folglich immer $1 \leq \gamma_P(G) \leq \gamma(G)$. Diese Abschätzung ist nach oben scharf, welches Abbildung 1.1 zeigt.

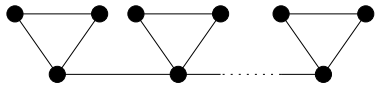


Abbildung 1.1: *In diesem Graph gilt* $\gamma_P(G) = \gamma(G)$,

Für beide Probleme gilt, dass ein Knoten aus jedem Dreieck in die entsprechende Menge übernommen werden muss. Folglich gilt hier $\gamma_P(G) = \gamma(G)$.
Es besteht aber auch die Möglichkeit, dass der Abstand zwischen $\gamma_P(G)$ und $\gamma(G)$ beliebig groß werden kann. Dies illustriert Abbildung 1.2. Die offenen Knoten bilden eine kleinstmögliche Dominating Set. Für den Pfad P_n der Länge n gilt $\gamma(P_n) = \lceil \frac{n}{3} \rceil$ und wie bereits in Beispiel 1.2 gesehen $\gamma_P(P_n) = 1$. Der viereckige Knoten u im Stern ist die kleinstmögliche Power Dominating Set. Je

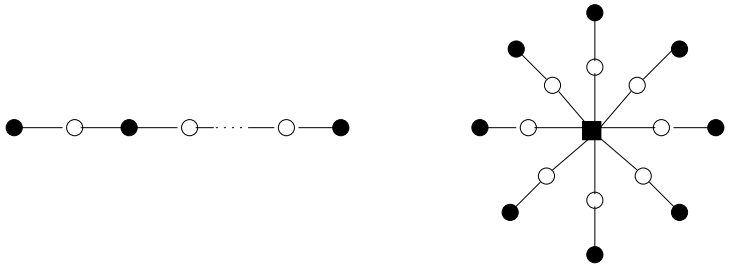


Abbildung 1.2: *Im Stern und im Pfad ist ein beliebig großer Abstand zwischen $\gamma(G)$ und $\gamma_P(G)$ möglich.*

größer der Stern wird, um so größer wird auch der Abstand zwischen $\gamma_P(G)$ und $\gamma(G)$. Für jeden Stern beträgt dieser Abstand $\gamma(G) - \gamma_P(G) = \delta(u) - 1$. Dies ist ebenfalls ein Beispiel dafür, dass für eine kleinstmögliche Power Dominating Set P und eine kleinstmögliche Dominating Set D für den gleichen Graphen nicht zwingend $P \subseteq D$ gelten muss.
Dominating Set ist bereits in großem Umfang untersucht worden und es liegen bereits viele weitergehende Resultate darüber vor. Hier stellt sich nun die Frage, ob und inwiefern sich solche Resultate auf Power Dominating Set übertragen lassen. Beispielsweise gibt es in G. Ausiello et al. [3] bereits für Dominating Set einen Approximationsalgorithmus mit einer Approximationsgüte von $\ln n$. Es wird auch gezeigt, dass die Approximation mit konstanter Approximationsgüte unmöglich ist, falls wir **NP≠P** annehmen. Ähnliche Resultate sind für Power Dominating Set bisher komplett unbekannt.
Im Gebiet der parametrisierten Komplexität zeigen R.G. Downey und M.R. Fellows [7], dass Dominating Set $W[2]$-hart ist, wenn man als Parameter die Größe der Dominating Set wählt. Es ist noch unbekannt, wie sich dies für Power Dominating Set verhält.
Ein weiterer interessanter Fokus ist das Gebiet der Datenreduktion. Für Dominating Set gibt es bereits ein sehr weitreichendes Resultat. In Alber et al. [1] wurden zwei Datenreduktionsregeln entwickelt, welche in polynomieller Zeit durchführbar sind. Die Autoren bewiesen, dass der Kern eines planaren Graphen, der Restgraph auf den keine Reduktionsregel mehr anwendbar ist, eine Größe besitzt, die linear abhängig ist von $\gamma(G)$. Dies zeigt, dass Dominating Set auf planaren Graphen in der parametrisierten Klasse **FPT** ist. Ein ähnliches Ergebnis für Power Dominating Set steht noch aus.

Kapitel 2

Domination und Power Domination

In [10] wird die **NP**-Vollständigkeit dadurch bewiesen, dass 3-SAT auf Power Dominating Set reduziert wird. Da 3-SAT kein graphentheoretisches Problem ist, lässt sich mit Hilfe der angeführten Reduktion nichts über die Komplexität von Power Dominating Set sagen, falls dieses auf spezielle Graphklassen beschränkt bleibt. Auch rein intuitiv würde man ebenfalls vermuten, dass sich Dominating Set auf Power Dominating Set reduzieren lässt.

2.1 Domination auf speziellen Graphklassen

Dominating Set ist auf allgemeinen Graphen **NP**-vollständig. Es liegen bereits viele komplexitätstheoretische Ergebnisse über eine große Anzahl von Graphklassen bezüglich Dominating Set vor. Die Tabelle 2.1 aus [17] gibt eine Übersicht über die wichtigsten Graphklassen und über die Komplexität von DS. In [10] wurde bereits für bipartite und chordale Graphen gezeigt, dass Power Dominating Set **NP**-vollständig ist. In diesem Abschnitt wollen wir **NP**-Vollständigkeit für weitere Graphklassen in Hinsicht auf Power Dominating Set zeigen. Ein wichtiges Instrument hierfür wird die nachfolgende Reduktion von Dominating Set auf Power Dominating Set sein. Unabhängig und nahezu zeitgleich wurde dieses Ergebnis bereits in J. Kneis et al. [15] veröffentlicht.

2.2 Reduktion von Domination auf Power Domination

Es soll im Folgenden eine Reduktion von Dominating Set auf Power Dominating Set konstruiert werden. Von nun an betrachten wir o.B.d.A. nur solche Graphen, die zusammenhängend sind. Wir dürfen auch davon ausgehen, dass es für G immer eine Power Dominating Set P gibt, sodass jeder Knoten in P Grad größer drei hat. In [10] wurde dies bereits erörtert. Wir wollen diese Erörterung ausführlich wiederholen.
Ist $\tilde{P}$ eine Power Dominating Set, welche oben genanntes Kriterium nicht erfüllt,

Graphklasse	Komplexitätsklasse
bipartite	**NP**
comparability	**NP**
chordal	**NP**
split	**NP**
planar	**NP**
circle	**NP**
interval	**P**
strongly chordal	**P**
dually chordal	**P**
permutation	**P**
cocomparabilty	**P**
AT-free	**P**
distance hereditary	**P**
k-polygon (festes $k \geq 3$)	**P**
partial k-tree (festes $k \geq 1$)	**P**

Tabelle 2.1: *Graphklassen und die Komplexität von* Dominating Set, *falls dieses auf die entsprechende Klasse beschränkt bleibt.* **P** *bedeutet dass es in polynomieller Zeit lösbar ist und* **NP** *weist auf die* **NP***-Vollständigkeit hin.*

so lässt sich auf einfache Art und Weise eine Lösung P gewinnen, sodass für alle $p \in P$ gilt $\delta(p) \geq 3$, falls $\Delta(G) \geq 3$. Sei $P_L = \{v \in V \mid \delta(v) \leq 2\} \cap \tilde{P}$. Für alle $v \in P_L$ gilt, dass hier zwei Pfade entspringen, die wir nun jeweils verfolgen. Auf mindestens einem dieser beiden Pfade treffen wir auf einen Knoten v' mit $\delta(v') \geq 3$. Entfernt man nun v aus $\tilde{P}$ und fügt v' hinzu, so bleibt G observiert. Nimmt man diese Ersetzung für alle Knoten in P_L vor, so resultiert eine pds P mit den gewünschten Eigenschaften. Da man für die Ersetzung für alle $v \in P_L$ höchstens n Knoten betrachten muss, geht dies in $O(n^2)$ Zeit.
Gilt $\Delta(G) < 3$, so ist G entweder ein Pfad oder ein Kreis, da G zusammenhängend ist. Dann ist Power Dominating Set trivial lösbar.
Die Reduktion geht folgendermaßen vor: Für jeden Knoten $u \in V$ wird ein neuer Knoten v_u erzeugt und über eine Kante mit u verbunden. Ein Graph $G(V, E)$ wird also auf einen Graphen $\hat{G}(\hat{V}, \hat{E})$ mit $\hat{V} = V \cup \{v_u \mid u \in V\}$ und $\hat{E} = E \cup \{\{u, v_u\} \mid u, v_u \in \hat{V}\}$ abgebildet. Ein Beispiel ist in Abbildung 2.1 gegeben.

Lemma 2.1. *D ist eine Dominating Set für $G(V, E)$ mit $|D| = k \iff$ D ist eine Power Dominating Set für $\hat{G}(\hat{V}, \hat{E})$ mit $|D| = k$.*

Beweis.
"$\Rightarrow$"
D ist eine pds für G, denn für alle $d \in D$ gilt, dass alle Knoten in $N[d]$ observiert sind. Da D eine Dominating Set ist, gibt es für alle $v \in V \setminus D$ ein $d \in D$ mit $v \in N[d]$. Somit sind alle Knoten in G durch Regel $\mathfrak{L}$ observiert. Um nur G mit D zu observieren braucht man die Regel $\mathfrak{G}$ nicht. Wir betrachten nun D als pds für $\hat{G}$. Der Graph G, als Teilgraph von $\hat{G}$, wird wieder nur durch Regel $\mathfrak{L}$ observiert. Übrig bleibt die Knotenmenge $\{v_u \mid u \in V\}$. Jeder Knoten v_u hat nur ein einzigen Nachbarn u. Für diesen gilt nach Konstruktion $N[u] \setminus \{v_u\} \subseteq V$. Da alle Knoten in V bereits observiert sind, kann nun v_u mit der Regel $\mathfrak{G}$

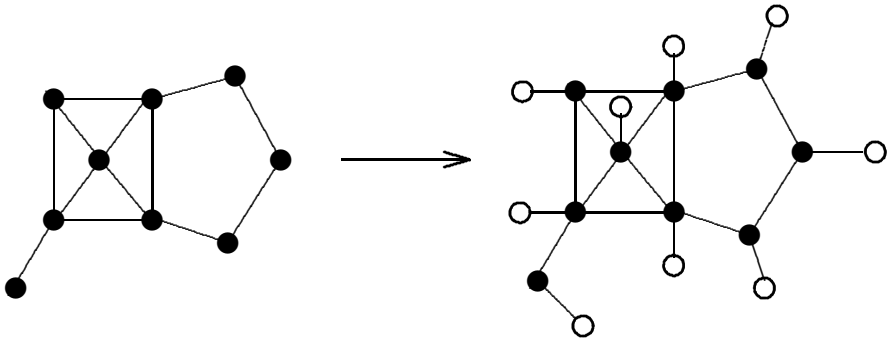


Abbildung 2.1: *Der Originalgraph G und der, welcher für die Reduktion von* Dominating Set *auf* Power Dominating Set *konstruiert wird.*

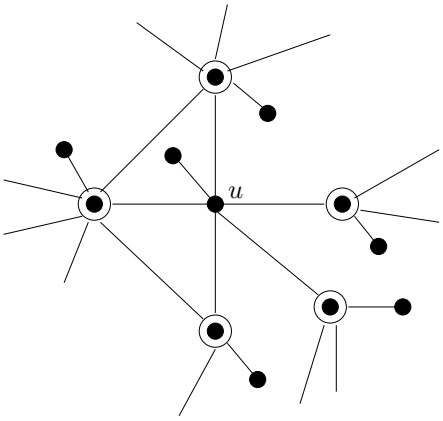


Abbildung 2.2: *Der Knoten u und $N_{\hat{G}}(u)$.*

observiert werden. Deswegen folgt, dass alle Knoten in $\{v_u \mid u \in V\}$ durch Regel $\mathcal{G}$ observiert werden können.

"$\Leftarrow$"
Nehmen wir nun an, D sei keine Dominating Set für G. Dann gibt es $u \in V$, so dass für alle $u' \in N_G[u]$ gilt $u' \notin D$. Wir betrachten nun u in $\hat{G}$ wie in Abbildung 2.2. Da für alle $d \in D$ gilt $\delta(d) \geq 3$, folgt dass alle Knoten $v_{u'}$ nur durch Regel $\mathcal{G}$ observiert werden. Dann muss insbesondere u schon vor allen $v_{u'}$ observiert sein. Dies ist aber nur möglich, falls für einen Nachbarn $\bar{u}$ zuvor $N[\bar{u}] \setminus \{u\}$ bereits observiert ist und somit u durch $\mathcal{G}$ observiert wird. Das heißt aber, dass $v_{\bar{u}}$ nicht mehr durch $\mathcal{G}$ observiert wird. Das kann nur $v_{\bar{u}} \in D$ bedeuten. Dies ist ein Widerspruch zur Wahl von D. □

Eine sofortige Konsequenz aus Lemma 2.1 und der in Garey und Johnson [9] nachgewiesenen **NP**-Vollständigkeit von Dominating Set auf allgemeinen Graphen, ist das nächste Theorem.

Theorem 2.2. Power Dominating Set *ist* **NP***-vollständig auf allgemeinen Graphen.*

2.3 Konsequenzen der Reduktion

Die konstruierte Reduktion von Dominating Set auf Power Dominating Set lässt sich aufgrund ihrer Beschaffenheit dazu benutzen weitere Ergebnisse für Power Dominating Set aus Dominating Set abzuleiten. Wir werden nun Aussagen über die Komplexität auf bestimmten Graphklassen machen, eine untere Schranke für die Approximation angeben und ein Ergebnis bezüglich der parametrisierten Komplexität erhalten.

2.3.1 Power Domination auf Graphklassen

Wir werden im Weiteren wie folgt vorgehen, um die **NP**-Vollständigkeit einer Graphklasse für Power Dominating Set zu folgern: Wir betrachten eine Graphklasse, die für Dominating Set **NP**-vollständig ist. Wir wenden auf jede Instanz dieser Graphklasse die Reduktion an. Sind die dadurch entstandenen Graphen ebenfalls in der gleichen Klasse, so muss nach Lemma 2.1 PDS auf dieser Graphklasse **NP**-vollständig sein. Für drei Klassen gehen wir im Folgenden noch genauer darauf ein.

Planare Graphen

Nach Garey und Johnson [9] ist Dominating Set sogar für planare Graphen mit Maximalgrad 3 **NP**-vollständig. Aus Lemma 2.1 folgt direkt:

Korollar 2.3. Power Dominating Set *ist* **NP***-vollständig für planare Graphen.*

Circle Graphen

Sei $G(V, E)$ ein Graph. Man sagt, dass G ein *Circle Graph* ist, falls eine Familie L von Sehnen auf einem Kreis existiert und diese Sehnen eine 1-zu-1-Beziehung zu den Knoten in V besitzen: Zwei Knoten sind genau dann zueinander adjazent falls sich ihre korrespondierenden Sehnen schneiden. Eine Familie L solcher Sehnen nennt man ein *Sehnenmodell*, siehe Abbildung 2.3.

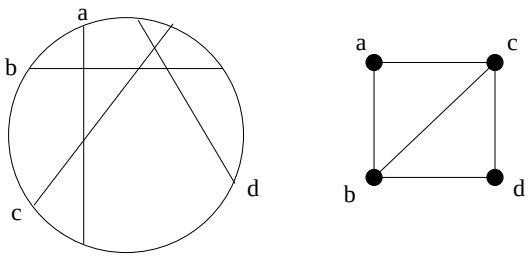


Abbildung 2.3: *Ein Circle Graph und sein Sehnenmodell.*

Wir können annehmen, dass sich keine zwei Sehnen auf dem Rand des Kreises schneiden. Denn trifft dieser Fall zu, so kann man durch die Vergrößerung des Kreisradius den Schnittpunkt ins Innere des Kreises verlagern.

Nach J.M. Keil [12] ist Dominating Set noch **NP**-vollständig, falls es auf Circle Graphen beschränkt bleibt.

Korollar 2.4. Power Dominating Set *ist auf der Klasse der Circle Graphen* **NP***-vollständig.*

Beweis. Wir müssen zeigen, dass, wenn die Reduktion aus Proposition 2.1 auf einen Circle Graphen G angewendet wird, der resultierende Graph ebenfalls ein Circle Graph ist. Dann ist PDS auf der Klasse der Circle Graphen **NP**-vollständig.
Wir besitzen nun ein Sehnenmodell mit Kreis R_1 für G. Da die Reduktion für jeden Knoten $u \in V$ einen neuen Knoten v_u erzeugt und die Kante $\{u_v, v\}$ einfügt, reicht es, zu zeigen, dass man für jede Sehne S eine weitere Sehne in den Kreis R_1 einfügen kann, die nur S schneidet. Wir können o.B.d.A. annehmen, dass keine Sehne einen Schnittpunkt mit einer anderen auf dem Rand von R_1 besitzt. Sei S eine Sehne und S_C ein Schnittpunkt mit dem Rand von R_1, dann kann man um S_C einen Kreis R_2 mit Radius $\epsilon > 0$ finden, sodass keine andere Sehne als S sich in ihm befindet. Der Rand von R_2 hat mit dem Rand von R_1 zwei Schnittpunkte. Verbinde nun diese Schnittpunkte. Dies ergibt eine Sehne in R_1, die nur S schneidet. □

Split Graphen

Ein Graph $G(V, E)$ ist ein *Split-Graph*, falls sich V in Knotenmengen V_1, V_2 partitionieren lässt, sodass der induzierte Subgraph $G(V_1)$ vollständig und $G(V_2)$ ohne Kanten ist. Klar ist, dass dann V_1 eine Clique bildet und V_2 eine unabhängige Menge (Independent Set) ist. Im Weiteren gehen wir o.B.d.A davon aus, dass V_1 immer eine maximale Clique bildet. Denn existiert ein $v \in V_2$, sodass $V_1 \cup \{v\}$ ebenfalls eine Clique ist, so ist dann $V_2 \setminus \{v\}$ in gleichem Maße ein Independent Set. Also ist $V_1 \cup \{v\}$ und $V_2 \setminus \{v\}$ ebenfalls eine Partition von V mit den geforderten Eigenschaften
In diesem Fall funktioniert die Reduktion aus Abschnitt 2.2 nicht, da der durch die Reduktion eines Split-Graphen entstandene Graph kein Split-Graph mehr ist. Dies sieht man so: Nach der Reduktion induziert $\hat{G}(V_1)$ immer noch eine maximale Clique. Für jeden Knoten $u \in V_2$ gilt, dass er nach der Reduktion zu einem neuen Knoten v_u von Grad eins adjazent ist. Da die Knoten in V_2 eine unabhängige Menge bilden müssen folgt $v_u \notin V_2$. In gleichem Maß kann auch v_u nicht zur Clique V_1 gehören, da v_u zu keinem Knoten aus V_1 adjazent ist. Der entstandene Graph lässt sich also nicht mehr wie gefordert partitionieren. Er kann also kein Split-Graph mehr sein.
Um die **NP**-Vollständigkeit von Power Dominating Set auf der Klasse der Split-Graphen zu beweisen, geben wir eine Reduktion von Vertex Cover auf Power Dominating Set an. Vertex Cover ist wie folgt definiert:

Vertex Cover
Eingabe: Ein Graph $G(V, E)$.
Ziel: Eine kleinstmögliche Menge $S \subseteq V$, so dass für alle $e \in E$ $e \cap S \neq \emptyset$ gilt.

Sei also ein Graph $G(V, E)$ gegeben. Erstelle zunächst den Inzidenzgraphen $I(W, F)$. Der Inzidenzgraph besteht aus dem Originalgraph und Knoten $V_E = \{v_e \mid e \in E\}$, welche die Kanten in E repräsentieren. Ein Knoten $v_e \in V_E$ wird

nun mit den beiden Knoten in V verbunden, zu denen die entsprechende Kante in $G(V, E)$ inzident ist. Also ist $W = V \cup V_E$ und $F = E \cup \{\{a, v_e\}, \{b, v_e\} \mid v_e \in V_E \text{ und } e = \{a, b\} \in E\}$. Als Zweites erweitern wir $I(W, F)$ zu $\tilde{I}(\tilde{W}, \tilde{F})$, indem wir für jeden Knoten $u \in V \subseteq W$ einen neuen Knoten v_u hinzufügen und diesen mit u verbinden. Es gilt also $\tilde{W} = W \cup T$ mit $T = \{v_u \mid u \in V \subseteq W\}$ und $\tilde{F} = F \cup \{\{v_u, u\} \mid v_u, u \in \tilde{W}\}$. Indem man den induzierten Subgraphen $\tilde{I}(V)$ zu einer Clique vervollständigt, erhält man H. H ist damit ein Split-Graph, denn $H(V)$ ist ein vollständiger Graph und $G(V_E \cup T)$ kantenlos. Abbildung 2.4 verdeutlicht sukzessive dieses Vorgehen.

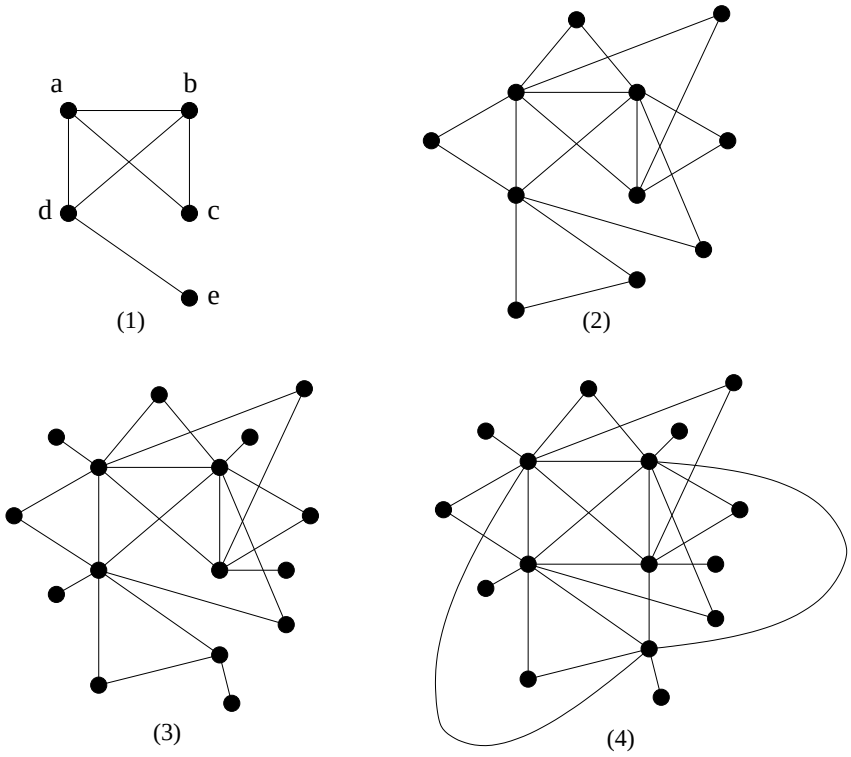


Abbildung 2.4: *(1)*$G(V, E)$ *(2)*$I(W, F)$ *(3)*$\tilde{I}(\tilde{W}, \tilde{F})$ *(4)*H.

Lemma 2.5. Power Dominating Set *ist für Split-Graphen* **NP***-vollständig.*

Beweis. Zu zeigen ist grundsätzlich, dass G genau dann ein Vertex Cover der Größe k hat, wenn H eine Power Dominating Set der Größe k hat.
Wir zeigen nun, dass wenn G ein Vertex Cover VC der Größe k hat, so ist VC ebenfalls eine Power Dominating Set für H. Da im Allgemeinen $k \geq 1$ gilt, ist $H(V)$ mit $\mathfrak{L}$ observiert. Da VC in G ein Vertex Cover ist, gilt für jede Kante $e = \{a, b\} \in E$, dass entweder a oder b (oder beide) in VC sind. Für alle $v_e \in V_E$ mit $e = \{a, b\}$ bedeutet dies, dass sie in H o.B.d.A. zu einem Knoten $a \in VC$ adjazent sind. Damit sind alle Knoten v_e ebenso mit $\mathfrak{L}$ observiert. Für alle Knoten $v_u \in T$ gilt entweder $u \in VC$, womit v_u durch $\mathfrak{L}$ observiert ist, oder aber es gilt $u \notin VC$ und $N[u] \setminus \{v_u\}$ ist nach Konstruktion mit $\mathfrak{L}$ observiert. Dies

ist so, da $H(V)$ bereits durch $\mathfrak{L}$ observiert ist und für alle v_e mit $e = \{u, l\} \in E$ gilt, dass $l \in VC$ ist. Dann ist v_u mit Anwendung von Regel $\mathcal{G}$ auf u observiert. Da nun ganz H observiert ist, ist VC eine Power Dominating Set.
Sei nun P eine Power Dominating Set für H und o.B.d.A. für alle $p \in P$ gelte $\delta(p) \geq 3$. Da wir wissen, dass für alle $v_e \in V_E$ $\delta(k_e) = 2$ und für alle $v_u \in T$ $\delta(v_u) = 1$ der Fall ist, folgt sofort $(V_E \cup T) \cap P = \emptyset$. Damit P nun ein Vertex Cover für G ist müssen wir zeigen, dass für alle v_e mit $e = \{a, b\} \in E$ entweder $a \in P$ oder $b \in P$ ist. Nehmen wir an $a, b \notin P$, dann kann v_e nur observiert werden, wenn o.B.d.A. Regel $\mathcal{G}$ auf a angewendet wird, siehe Abbildung 2.5. Dies heißt, dass insbesondere v_a zuvor observiert sein muss und zwar weder dadurch, dass $a \in P$ gilt noch dadurch, dass $\mathcal{G}$ auf a angewendet wurde. Dies lässt aber nur den Schluss $v_a \in P$ zu, welches ein Widerspruch zur Wahl von P ist. Also ist P Vertex Cover für G. □

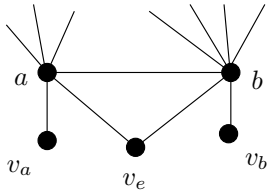


Abbildung 2.5: *Die Nachbarschaft zweier Knoten $a, b \in V$ in H.*

Wir konnten nun für einige Graphklassen die **NP**-Vollständigkeit bezüglich PDS zeigen. Alle Graphklassen die in Tabelle 2.1 für Dominating Set **NP**-vollständig sind, sind dies ebenfalls für Power Dominating Set. Wir wollen der Tabelle 2.1 eine weitere Spalte für Power Dominating Set hinzufügen. Damit erhalten wir Tabelle 2.2. Die **NP**-Vollständigkeit der Klasse der Comparability-Graphen folgt aus der **NP**-Vollständigkeit der bipartiten Graphen. Dies kann man folgern, da die Klasse der bipartiten Graphen eine Teilmenge der Klasse der Comparability-Graphen bildet.
In diesem Abschnitt konnten wir also zeigen, was uns unsere Intuition bereits angedeutet hat. PDS ist mindestens so schwer wie DS. Auf allen betrachteten Graphklassen, für die DS **NP**-vollständig ist, ist es auch PDS. Im nächsten Kapitel wollen wir uns einigen Graphklassen zuwenden, für die Domination in **P** ist. Wir wollen dann für Power Domination selbiges zeigen.

2.3.2 Grenze der Approximierbarkeit

In [3] wird von G. Ausiello et al. gezeigt, dass eine bessere Approximationsgüte als $\log n$ für Dominating Set nicht möglich ist, falls wir $\mathbf{P} \neq \mathbf{NP}$ annehmen. Hiermit und mittels Theorem 2.1 folgern wir Ähnliches für Power Dominating Set.

Korollar 2.6. *Falls $\mathbf{P} \neq \mathbf{NP}$ gilt, gibt es für* Power Dominating Set *keinen Algorithmus mit besserer Approximationsgüte als* $\log n - \log 2$.

Beweis. Angenommen es würde ein Algorithmus mit einer besseren Approximationsgüte existieren. Sei ein beliebiger Graph $G(V, E)$ gegeben. Wir wenden nun die Reduktion aus Abschnitt 2.2 von Dominating Set nach Power Dominating Set auf G an. Es entsteht $\hat{G}(\hat{V}, \hat{E})$. Sei L_{Opt} die optimale Lösung

Graphklasse	Komplexitätsklasse	
	Dominating Set	Power Dominating Set
bipartite	**NP**	**NP**
comparability	**NP**	**NP**
chordal	**NP**	**NP**
split	**NP**	**NP**
planar	**NP**	**NP**
circle	**NP**	**NP**
interval	**P**	?
strongly chordal	**P**	?
dually chordal	**P**	?
permutation	**P**	?
cocomparabilty	**P**	?
AT-free	**P**	?
distance hereditary	**P**	?
k-polygon (festes $k \geq 3$)	**P**	?
partial k-tree (festes $k \geq 1$)	**P**	?

Tabelle 2.2: *Graphklassen und die Komplexität von* Dominating Set *und* Power Dominating Set*, falls diese auf die entsprechende Klasse beschränkt bleiben.* **P** *bedeutet, dass das Problem in polynomieller Zeit lösbar ist und* **NP** *weist auf die* **NP***-Vollständigkeit hin.*

und L_{App} die Lösung dieses Algorithmus für $\hat{G}$. Mit $n' := |\hat{V}|$, $n := |V|$ gilt $n' = 2n$ und $\frac{L_{App}}{L_{Opt}} < \log|\hat{V}| - \log 2 = \log n' - \log 2$. Dann sind aber L_{App} und L_{Opt} nach Theorem 2.1 auch Lösungen für G. Insbesondere ist L_{Opt} auch eine optimale Lösung für G. Es folgt:

$$\frac{L_{App}}{L_{Opt}} < \log n' - \log 2 = \log 2n - \log 2 = \log 2 + \log n - \log 2 = \log n.$$

Dies ist aber ein Widerspruch zur Approximationsgüte von Dominating Set in [3]. □

Korollar 2.6 scheint nun ein Hinweis darauf zu sein, dass PDS bezüglich Approximierbarkeit mindestens so schwierig ist wie DS. Ein weiterer interessanter Aspekt wäre die Approximierbarkeit von Power Domination auf bestimmten Graphklassen.

2.3.3 Parametrisierte Komplexität

Aufgrund der speziellen Struktur der Reduktion aus Abschnitt 2.2 können wir ein Ergebnis bezüglich der parametrisierten Komplexität von Power Dominating Set mit Hilfe von Dominating Set gewinnen.

Korollar 2.7. Power Dominating Set *ist $W[2]$-hart, wenn wir die Größe der Power Dominating Set als Parameter wählen.*

Beweis. In [17] wird gezeigt, dass Dominating Set $W[2]$-hart ist, falls die Größe der Dominating Set als Parameter k gewählt wird. Wir können dieses

Ergebnis mit Hilfe der Reduktion aus Abschnitt 2.2 auf Power Dominating Set übertragen. Dies ist möglich, da es sich um eine parametrisierte Reduktion handelt. Denn sei ein Graph $G(V, E)$ gegeben, dann ist mit Lemma 2.1 (x, k) mit $x \subseteq V$ und $k = |x|$ genau dann eine Lösung der Größe k von G für DS, wenn (x', k') mit $x' = x$ und $k' = k$ eine Lösung der Größe k' für DS von $\hat{G}$, welcher der durch Reduktion erhaltenen Graphen ist, für PDS ist. Da die Reduktion zusätzlich in Zeit $O(n)$ durchführbar ist, ist sie eine parametrisierte Reduktion. □

Kapitel 3

Power Domination auf Bäumen

Im folgenden Kapitel wird ein Linearzeit-Algorithmus zur Berechnung einer kleinstmöglichen pds für Bäume vorgestellt. In [10] wurde bereits ein solcher Algorithmus präsentiert. Doch ist dieser sehr kompliziert geartet und besteht aus vielen speziellen Fallunterscheidungen. Der hier präsentierte Algorithmus besitzt einen schematischeren Zugang für den Leser. Der Algorithmus verfolgt einen Bottom-Up-Ansatz. Ausgehend von den Blättern, die am weitesten entfernt von der Wurzel sind, geht man schichtweise den Baum nach oben. Dabei versucht man so spät wie möglich einen PMU zu setzen, sodass es in den bereits betrachteten unteren Schichten keine unobservierten Bereiche gibt.

3.1 Einteilung in Schichten

Definition 3.1. *Ein* einfacher Pfad *ist ein Pfad* $u, v_1 \dots v_n, u'$*, sodass für alle* v_i*,* $1 \leq i \leq n$*,* $\delta(v_i) = 2$ *gilt.*

Beobachtung 3.2. *Sei* $P = u, v_1 \dots v_n, u'$ *ein einfacher Pfad. Ist entweder* u *oder* u' *observierend, so ist ganz* P *observiert.*

Beweis. Sei o.B.d.A u observierend. Aus Beispiel 1.2 wissen wir, dass sich dann die Observation entlang P mit Regel $\mathcal{G}$ ausbreitet. Da $\delta(u') \geq 1$ kann allerdings auf dem Knoten u' nicht zwingend Regel $\mathcal{G}$ angewendet werden. Also ist P observiert. □

Es reicht also entweder u oder u' in eine pds aufzunehmen, um P und zusätzlich noch die Knoten aus der entsprechenden Nachbarschaft von u bzw. u' zu observieren.
Man kann also sagen, dass Knoten mit Grad größer zwei grundsätzlich als Barrieren bezüglich der Fortpflanzung von Observation angesehen werden können. Ein weiteres wichtiges Faktum wurde bereits in Kapitel 2 angesprochen: Für einen Graphen G mit $\Delta(G) \geq 3$ gibt es eine kleinstmögliche pds in der kein Knoten mit Grad kleiner als 3 enthalten ist.
Sei T ein Baum $T = (V, E)$ mit Wurzel r. Gilt für zwei Knoten u und v, dass u auf dem Pfad von r nach v liegt, so wird v im weiteren als *tieferstehend* zu

u bezeichnet. Wenn wir davon sprechen, dass ein Pfad $P = v_1, \ldots, v_k$, welcher in einem Baum verläuft, in den Knoten v_k *von unten mündet*, so bedeutet dies, dass alle Knoten v_i für $1 \leq i < k$ jeweils tieferstehend zu v_{i+1} sind. Wir teilen die Knotenmenge V in zwei disjunkte Teilmengen V_H und V_L ein:

$$V_L := \{v \in V \mid \delta(v) \leq 2\}$$
$$V_H := \{v \in V \mid \delta(v) > 2\}$$

Offensichtlich gilt $V = V_H \dot{\cup} V_L$. Der Algorithmus geht konzeptuell so vor: Nachdem man ein Blatt als Wurzel r festgelegt hat, teilt man, von r ausgehend, die Knoten in V_H in Schichten L_i für $i \geq 1$ ein, wobei

$$L_i := \{v \in V_H \mid \exists P_v = r, \ldots, v \ , \ |P_v \cap V_H| = i\}.$$

Gilt also $v \in L_i$, so hat der Pfad von r nach v exakt i Knoten aus V_H. Ist n der Index der tiefsten Schicht, so gilt $V_H = \dot{\bigcup}_{i=1\ldots n} L_i$. Die Schichten werden dann nacheinander, angefangen bei der tiefsten, abgearbeitet. Für jeden Knoten werden die von unten eingehenden Pfade betrachtet. Zum Beispiel: Bei einem Knoten u aus der tiefsten Schicht L_n sind dies nur einfache Pfade zwischen u und den Blättern, welche Nachfahren von u sind. Sei $k > 1$ die Anzahl dieser Pfade. Nach Beobachtung 3.2 reicht es, wenn u observierend ist, um diese Pfade zu observieren. Man könnte sie auch dadurch observieren, dass man Regel $\mathcal{G}$ auf u anwendet. Da aber $\delta(u) = k + 1$ gilt, müssten hierfür k Knoten observierend sein, also mindestens 2 Knoten. Es gibt demnach keine bessere Lösung als auf u eine PMU zu setzen. Daher folgt:

Beobachtung 3.3. *Ist $u \in V_H$ ein Knoten aus der untersten Schicht L_n, dann gibt es eine kleinstmögliche Power Dominating Set S mit $u \in S$.*

Allerdings propagiert nun u mit Regel $\mathcal{G}$ Observation auf dem Pfad zur Wurzel den Baum nach oben. Dies soll heißen, dass wir nun $\mathcal{G}$ so lange wie möglich anwenden. Diese Observation pflanzt sich fort bis der Pfad von u zur Wurzel auf einen Knoten $z \in V_H$ trifft. Der Knoten z gehört dann zur Schicht L_{n-1}. Der Algorithmus unterscheidet alle von einem $z' \in L_n$ oder einem Blatt ausgehenden einfachen Pfade, die von unten in z münden, bezüglich ihrer Observation und aufbauend hierauf entscheidet er, ob z observierend sein muss. Dazu müssen natürlich aber erst alle Knoten aus L_n abgearbeitet sein.
Für $u \in V$ meinen wir mit T_u im Weiteren den Teilbaum, der durch u und alle Knoten, die tieferstehend zu u sind, induziert wird.

3.2 Pfadtypen

Definition 3.4. *Sei $P = u, v_1 \ldots v_n, u'$ ein einfacher Pfad in einem Baum und es gilt $u' \in L_k$ und $u \in L_{k-1}$.*

a) Falls $u, v_1 \ldots v_n$ unobserviert ist und aus der Observation von P folgt, dass $T_{u'}$ observiert ist, so ist P ein Typ-1-Pfad*, siehe Abbildung 3.1.*

b) P ist ein Typ-2-Pfad*, falls er ein Typ1-Pfad ist und zusätzlich gilt, dass es reicht u zu observieren um P und $T_{u'}$ zu observieren, siehe Abbildung 3.2.*

c) P ist ein Typ-3-Pfad*, falls P und $T_{u'}$ bereits observiert sind.*

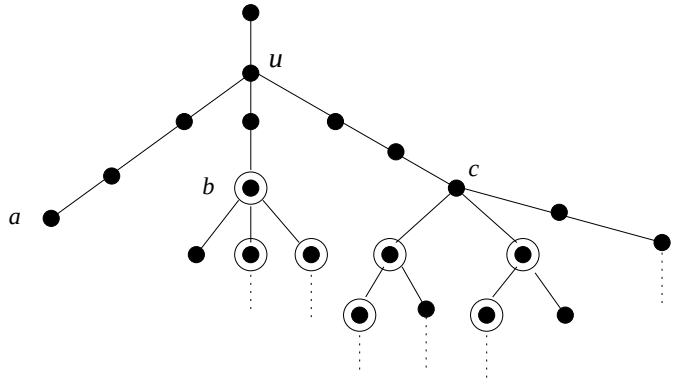


Abbildung 3.1: *Mehrere beispielhafte Typ-1-Pfade. Die Knoten a, b, c stellen u' in $P = u, v_1 \dots v_n, u'$ dar.*

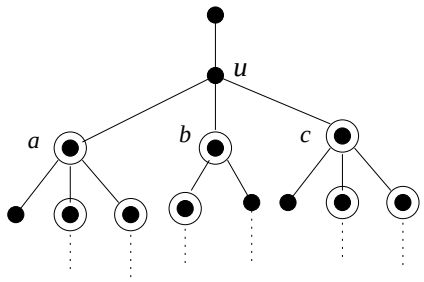


Abbildung 3.2: *Mehrere beispielhafte Typ-2-Pfade. Die Knoten a, b, c stellen u' in $P = u, v_1 \dots v_n, u'$ dar.*

Mit Definition 3.4 ergibt sich sofort folgende Beobachtung:

Beobachtung 3.5. *Sobald an einem Knoten $u \in V$ ein Typ-3-Pfad von unten mündet, kann kein Typ-2-Pfad mehr von unten in u münden.*

Beweis. Der Knoten u ist bereits observiert und somit wäre ein potentieller Typ-2-Pfad mit seinem Unterbaum ebenso observiert. □

Der Algorithmus setzt in seinem Verlauf die PMU's gerade so, dass nur Pfade vom Typ-1, Typ-2 und Typ-3 entstehen. Dadurch ist es unmöglich, dass es Teile im Baum gibt, die, ausgehend von der aktuellen Schicht, diejenige welche der Algorithmus momentan betrachtet, nicht mehr observierbar sind. Um dies zu sehen brauchen wir noch die nächste Aussage.

Lemma 3.6. *Sei $u \in L_{i-1}$, $u' \in L_i$ und sei in $T_{u'}$ eine observierende Knotenmenge vorhanden, so dass es nur Typ-1, Typ-2 und Typ-3-Pfade gibt, die von unten in u' münden. Sei $P = u, v_1 \dots v_n, u'$ ein einfacher Pfad der von unten in u mündet, dann gilt*

1. *P ist ein Typ-2-Pfad $\iff$ $P = u, u'$ und es gilt, dass in u' $\delta(u') - 2$ Typ-3-Pfade und ein Typ-1-Pfad von unten münden.*

2. *P ist ein Typ-1-Pfad $\iff$ in u' münden entweder*
   a) $\delta(u') - 2$ Typ-3-Pfade und ein Typ-1-Pfad und es gilt $P \neq u, u'$ oder
   b) mindestens $\delta(u') - 2$ Typ-2-Pfade und kein Typ-3-Pfad münden von unten in u'.

Beweis.

1. "$\Longrightarrow$"
   Da es reicht u zu observieren, damit P observiert ist, unabhängig davon ob ein weiterer unobservierter Typ-1-Pfad in u mündet, kann Regel $\mathcal{G}$ nicht zwingend auf u angewendet werden. Wenn P aber dadurch, dass u observiert ist, ebenfalls observiert ist, dann gilt $P = u, u'$. Es muss u' zuvor schon observiert gewesen sein, da u keinen Einfluss mittels $\mathcal{G}$ auf u' hat. Also muss mindestens ein Typ-3-Pfad in u' von unten münden und mit Beobachtung 3.5 kann kein Typ-2-Pfad in u' von unten münden.
   Angenommen es münden $\delta(u') - 2 - k$, $k \geq 1$, viele Typ-3-Pfade von unten in u', dann münden auch $k + 1 > 1$ viele Typ-1-Pfade von unten in u'. Ist P observiert, kann auf u' Regel $\mathcal{G}$ nicht angewendet werden. Somit bleiben die Typ-1-Pfade unobserviert und $T_{u'}$ ist nicht vollständig observiert. Widerspruch.
   Angenommen es münde $\delta(u') - 1$ viele Typ-3-Pfade von unten, dann ist mit Regel $\mathcal{G}$ auch P observiert und somit ein Typ-3-Pfad. Widerspruch.
   "$\Longleftarrow$"
   Da $\delta(u') \geq 3$ gilt, gibt es mindestens einen Typ-3-Pfad, der von unten in u' mündet. Damit ist u' observiert. Wird nun u observiert, so kann auf u' Regel $\mathcal{G}$ angewendet werden, sodass der einzige Typ-1-Pfad observiert wird. Dann ist aber ganz $T_{u'}$ observiert.

2. "$\Longrightarrow$"
   Entweder ist u' observiert oder unobserviert. Im ersten Fall muss ein Typ-3-Pfad von unten in u' münden. Deshalb kann mit Beobachtung 3.5 u' auf keinem Typ-2-Pfad liegen. Der Knoten u' muss ein unobserviertes Kind besitzen, da sonst mit der Anwendung von $\mathcal{G}$ auf u' ganz P observiert wäre. Also muss in u' ein Typ-1-Pfad von unten münden, welcher auch das unobservierte Kind enthält. Die Fälle in denen $\delta(u') - 2 - k$, $k \geq 1$, bzw. $\delta(u') - 1$ viele Typ-3-Pfade in u' von unten münden, führen analog wie in 1 zu Widersprüchen. Dies zeigt a).
   Sei u' nun unobserviert. Also kann kein Typ-3-Pfad von unten in u' münden. Es kann höchstens ein Typ-1-Pfad in u' münden, da mehr als einer von u aus nicht mehr observierbar ist. Dann ist aber klar, dass die restlichen Pfade, die von unten münden, Typ-2-Pfade sein müssen. Da u' auf P und eventuell auf einem Typ-1-Pfad liegt, müssen dies mindestens $\delta(u') - 2$ viele Typ-2-Pfade sein. Dies zeigt b).
   "$\Longleftarrow$"

   (a) In u münden $\delta(u') - 2$ Typ-3-Pfade und ein Typ-1-Pfad. Wird P observiert, so kann auf u' Regel $\mathcal{G}$ angewendet werden. Der Typ-1-Pfad ist dann observiert und somit ebenso $T_{u'}$.

   (b) Es münden $\delta(u') - 2$ viele Typ-2-Pfade von unten in u'. Da kein Typ-3-Pfad vorkommen kann, muss noch ein Typ-1-Pfad in u' von unten münden. Wird P observiert, so auch u'. Dann sind aber auch alle

Typ-2-Pfade observiert und mit Regel $\mathcal{G}$ auf u' auch der Typ-1-Pfad. Münden $\delta(u')-1$ viele Typ-2-Pfade von unten in u', sind diese ebenso alle observiert, wenn P observiert wird. Also folgt, dass wenn P observiert ist, so ist ebenfalls $T_{u'}$ observiert. □

Wir wissen nun einiges über die Typ-1 und Typ-2-Pfade zwischen Knoten in benachbarten Schichten. Wenn ein Pfad nicht die Voraussetzung für einen Typ-1 oder Typ-2-Pfad erfüllt, muss eine PMU so gesetzt werden, dass er ein Typ-3-Pfad wird. Wir haben in Lemma 3.6 nur Pfade zwischen zwei Schichten L_i und L_{i+1} betrachtet. Nun richten wir das Augenmerk auf die Pfade, die in den Blättern beginnen.

Beobachtung 3.7. *Sei z ein Blatt, $u \in L_i$ und $P = z, v_1 \dots v_n, u$ ein einfacher Pfad, der in u von unten mündet. Dann ist P ein Typ-1-Pfad.*

Beweis. Ist P observiert, dann auch $T_z = G(\{z\}, \emptyset)$. Also ist P Typ-1-Pfad. □

3.3 Vorgehen des Algorithmus

Lemma 3.6 gibt uns bereits an, wann ein einfacher Pfad $P = u, v_1 \dots v_n, u'$ zwischen zwei Knoten $u, \in L_i$, $u' \in L_{i+1}$ in Abhängigkeit von den Pfadtypen an u' ein Typ-1 bzw. Typ-2-Pfad ist. Wir müssen nun alle Möglichkeiten, in denen die drei Pfadtypen in Kombination vorkommen können, betrachten. Lemma 3.8 wird uns Handlungsweisen mitgeben, um zu entscheiden in welchen Fällen der Pfad P welchen Typ hat und wann u in die pds genommen werden muss. Zu jedem Zeitpunkt soll gelten, dass es nur Pfade von beschriebenem Typ gibt und keine unobservierten Teile existieren, die nicht von einem Knoten oberhalb von u observierbar sind.

3.3.1 Observation einer Schicht

Sei $\text{Typ}_i(v)$ die Anzahl der Pfade vom Typ i, die in v von unten münden.

Lemma 3.8. *Sei $u \in L_i$, $u' \in L_{i+1}$ und $P = u, v_1 \dots v_n, u'$ ein einfacher Pfad. Seien in $T_{u'}$ die PMU's so gesetzt, dass es dort nur Typ-1, Typ-2 und Typ-3-Pfade gibt. Dann muss folgendes gelten, damit in T_u ebenfalls nur die genannten Pfade vorkommen.*

1. $\text{Typ}_1(u') \geq 2 \Rightarrow u' \in$ *pds und P ist Typ-3-Pfad.*
2. $\text{Typ}_2(u') \geq 1$, $\text{Typ}_1(u') \leq 1$, $\text{Typ}_3(u') = 0 \Rightarrow P$ *ist Typ-1-Pfad.*
3. $\text{Typ}_1(u') = 1$, $\text{Typ}_3(u') \geq 1$ *und* $n = 0 \Rightarrow P$ *ist Typ-2-Pfad.*
4. $\text{Typ}_1(u') = 1$, $\text{Typ}_3(u') \geq 1$ *und* $n > 1 \Rightarrow P$ *ist Typ-1-Pfad.*
5. $\text{Typ}_1(u') = 0$ *und* $\text{Typ}_3(u') \geq 2 \Rightarrow P$ *ist Typ-3-Pfad.*

Beweis.

1. Nach Lemma 3.6 kann P weder ein Typ-1 noch ein Typ-2-Pfad sein. Es muss also die pds erweitert werden, damit P ein Typ-3-Pfad wird. Dazu muss jeder Typ-1-Pfad, der von unten in u' mündet, und P observiert werden. Pro Pfad muss ein Knoten observierend sein. Da u' zu all diesen Pfaden gehört, reicht es aus u' in die pds zu nehmen. Dies ist analog zur Beobachtung 3.3 dann auch die kleinste Anzahl Knoten, die in die pds genommen werden muss. Infolgedessen werden P und $T_{u'}$ observiert.
2. Lemma 3.6.2 b).
3. Lemma 3.6.1.
4. Lemma 3.6.2 a).
5. Auf u' kann Regel $\mathcal{G}$ angewendet werden und P ist observiert. Da alle in u' von unten mündenden Pfade vom Typ-3 waren, ist $T_{u'}$ observiert. Also ist P auch ein Typ-3-Pfad. □

Weitere Fälle können nicht auftauchen. Denn zählt man die Möglichkeiten in denen die drei Pfadtypen miteinander auftreten können, sind dies sieben. Da sich Typ-2 und Typ-3-Pfade jeweils ausschließen, entfallen zwei dieser Möglichkeiten. Da wir zwischen dem Auftreten von einem und mehreren Typ-1-Pfadtypen unterscheiden, müssen die Fälle, in denen sie mit Typ-2 bzw. Typ-3-Pfaden auftauchen, doppelt gezählt werden. Es kommen also noch zwei hinzu, insgesamt also sieben. Überprüfen wir, ob die fünf Fälle in Lemma 3.8 dies abdecken. Der 1. Fall deckt drei Möglichkeiten, der 2. Fall zwei Möglichkeiten, der 3. und 4. Fall zusammen eine und der 5. Fall wiederum eine ab. Dies sind zusammen ebenfalls sieben Möglichkeiten. In Fall 3 und 4 haben wir noch zwischen der Länge des Pfades P unterschieden. Für Fall 1 und 5 macht dies keinen Sinn, da hier P Observation den Baum nach oben propagiert. Ebenso müssen wir diese Unterscheidung in Fall 2 nicht treffen, da hier u' nicht observiert ist. Eine Übersicht hierüber ist in Tabelle 3.1 gegeben.

3.3.2 Traversierung der Schichten

Lemma 3.8 liefert die grundsätzliche Idee für einen Algorithmus zur Berechnung einer kleinstmöglichen pds für einen Baum. Der vollständige Algorithmus ist in Algorithmus 1 zu sehen.

Um die Knoten in V_H in Schichten L_i einteilen zu können, verwenden wir eine Variante der Breitensuche. Bei der gewöhnlichen Breitensuche werden alle Knoten mit Abstand k zur Wurzel vor jedem Knoten mit Abstand $k+1$ oder größer entdeckt. In unserer Variante werden einfache Pfade $u, p_1 \dots p_n, v$, mit $\delta(u) > 2$, $\delta(v) > 2$ und $\delta(p_i) = 2$ für alle $1 \leq i \leq n$, als Kante $\{u, v\}$ betrachtet. Hieraus folgt nun, dass alle Knoten aus V_H, die über einen Pfad mit k Knoten aus V_H von r aus erreichbar sind, vor denen gefunden werden, die über einen Pfad mit $k+1$ oder mehr Knoten aus V_H erreichbar sind. Jedes Mal wenn wir im Verlauf auf solch einen Knoten aus V_H treffen, wird er vorne in eine separate Liste gestellt. Wenn wir nun diese Liste später von vorne abarbeiten, treffen wir immer auf Knoten in L_i bevor wir auf welche aus L_{i-1} treffen.

Algorithm 1 *Berechnet eine kleinstmögliche pds für einen Baum*

Eingabe: Ein Baum $G(V, E)$.

Ausgabe: Eine Menge $P \subseteq V$, so dass jeder Knoten observiert und $|P|$ minimal ist.

1. Lege ein Blatt als Wurzel r des Baumes fest.
2. Breitensuchvariante: Jedes Mal wenn ein Knoten v mit $\delta(v) > 2$ gefunden wird, füge ihn vorne in Liste L ein. Füge alle Blätter in eine Liste U ein.
3. $P \leftarrow \emptyset$
4. **for all** $u \in U$ **do**
   follow(u,1)
   end for
5. 1: **while** L $\neq \emptyset$ **do**
   2: $u \leftarrow$ L.pop
   3: **if** typecount[u,1]$\geq$ 2 **then** {1. Fall}
   4: $P \leftarrow P \cup \{u\}$
   5: follow(u,3)
   6: **end if**
   7: **if** typecount[u,2]$\geq$1 **and** typecount[u,3]=0 **and** typecount[u,1]$\leq$1 **then** {2. Fall}
   8: follow(u,1)
   9: **end if**
   10: **if** typecount[u,3]$\geq$2 **and** typecount[u,1]=0 **then** {5. Fall}
   11: follow(u,3)
   12: **end if**
   13: **if** typecount[u,3]$\geq$2 **and** typecount[u,1]=1 **then** {3. und 4. Fall}
   14: **if** $\delta(parent(u)) > 2$ **then** {4. Fall}
   15: follow(u,2)
   16: **else** {3. Fall}
   17: follow(u,1)
   18: **end if**
   19: **end if**
   20: **end while**

Prozedur:
follow(v,typ):

1: **repeat**
2: $v \leftarrow$ parent(v)
3: **until** $\delta(v) > 2$ **or** parent(v) = $\emptyset$
4: typecount[v,typ]$\leftarrow$typecount[v,typ]+1
5: **if** parent(v) = $\emptyset$ **and** (typ=1 **or** typ=2) **then**
6: $P \leftarrow P \cup \{v\}$
7: **end if**

No.	Typ-1		Typ-2	Typ-3	Fall
	#Typ-1=1	#Typ-1>1			
1	-	-	⋆	-	3.8.2
2	-	-	-	⋆	3.8.5
3	-	-	⋆	⋆	äquivalent zu No. 2
4	⋆	-	⋆	-	3.8.2
5	⋆	-	-	⋆	$n=0$ $n>0$ 3.8.3 3.8.4
6	⋆	-	⋆	⋆	äquivalent zu No. 5
7	-	⋆	-	-	3.8.1
8	-	⋆	⋆	-	3.8.1
9	-	⋆	-	⋆	3.8.1
10	-	⋆	⋆	⋆	äquivalent zu No. 9

Tabelle 3.1: *Die möglichen Kombinationen der Pfadtypen. Ein "⋆"-Zeichen bedeutet, dass der entsprechende Pfad vorliegt, ein "-"-Zeichen, dass er fehlt.*

Für jeden Knoten v in einer Schicht L_i werden die drei Pfadtypen in einem Array typecount$[v, i]$, welcher mit v und $i \in \{1, 2, 3\}$ indiziert wird, gezählt. Mit der Prozedur parent() wird der eindeutige Elternknoten eines beliebigen Knotens in einem Baum ermittelt. Die Prozedur follow(v,typ) verfolgt den einfachen Pfad von v aus den Baum nach oben bis zu einem Knoten mit Grad größer als zwei in der direkt über v liegenden Schicht. Dort inkrementiert sie abhängig von dem Parameter typ den entsprechenden Zähler für die Pfadtypen mit dem Befehl "typecount[v,typ] ← typecount[v,typ]+1". Falls aber typ=1 oder typ=2 und $v = r$ gilt, so wird v in die pds hinzugenommen, da ein Typ-1 bzw. Typ-2-Pfad noch observiert werden muss. In Punkt 5 wird jeder Fall aus Lemma 3.8 abgedeckt. Die Kommentare geben genau an welcher Fall in welcher Zeile behandelt wird.
Von der tiefsten Schicht beginnend, betrachtet der Algorithmus jeden Knoten in den direkt aufeinander folgenden Schichten. Für jeden Knoten trifft einer der 5 Fälle zu und jedes Mal führen wir dann die entsprechenden Aktionen durch. Jede dieser Aktionen versucht immer nur einen Knoten in die pds zu nehmen, falls nötig. Nur dann also, wenn es unmöglich ist, dass von einen Knoten in einer höheren Schicht Observation zum aktuellen Knoten gelangen kann.

3.4 Korrektheit und Laufzeit

Für ein $v \in V_H$ sei $parent_{V_H}(v)$ der nächste Knoten aus V_H wenn man von v aus Richtung Wurzel läuft.

Beobachtung 3.9. *Nimmt der Algorithmus 1 einen Knoten $u \in L_i$ in die pds auf, so kann stattdessen $parent_{V_H}(u) \in L_{i-1}$ nicht hinzugenommen werden, ohne dass T_u teilweise nicht observiert wäre.*

Beweis. Der Algorithmus fügt einen Knoten in die pds, wenn in u mehr als ein Typ-1-Pfad mündet. Wird $parent_{V_H}(u)$ stattdessen in die pds aufgenommen, ist zwar der Pfad von $parent_{V_H}(u)$ nach u observiert. Da aber mehr als ein Typ-1-

Pfad in u von unten mündet kann Regel $\mathcal{G}$ nicht auf u angewandt werden. Also bleiben die Typ-1-Pfade unobserviert. □

Proposition 3.10. *Algorithmus 1 berechnet eine optimale pds für einen Baum.*

Beweis. Sei $M \subseteq V$ die von Algorithmus 1 berechnete Menge für den Baumgraphen $G(V, E)$. Als erstes müssen wir zeigen, dass M eine pds ist. Für alle u aus der Liste L wissen wir, dass im Laufe des Algorithmus observierte und unobservierte Pfade dort von unten münden. In Punkt 4 des Algorithmus werden die Pfade von einem Blatt zu einem Knoten aus L_n nach Beobachtung 3.7 als Typ-1-Pfade gekennzeichnet. Mit Lemma 3.8 wissen wir, dass es reicht die unobservierten Pfade zu observieren, damit der Teilbaum T_u unter dem aktuellen Knoten u observiert ist. Liegen nur observierte Pfade vor oder nehmen wir u in die pds, so ist T_u observiert. Es reicht sonst den Pfad P, der von u aus zur nächsthöheren Schicht führt, zu observieren, damit T_u observiert ist. Sei P' der Pfad von u nach r. Für alle $v \in (P' \cap V_H) \setminus \{u\}$ gilt ebenfalls das bereits erwähnte. Also wird entweder v der pds hinzugefügt und Observation wird von v nach u über P' propagiert und P ist damit observiert, da $P \subseteq P'$ gilt. Oder aber r kommt in die pds und das gleiche geschieht. Dies gilt insbesondere für den Knoten in L_1. Also ist M eine pds.
Nehmen wir an, es gibt $M' \subseteq V$, M' ist eine optimale pds für G, für alle $m \in M'$ gilt $\delta(m) > 2$ und es gilt $|M'| < |M|$. Seien $L_1, \ldots, L_n$ die Schichten in die V_H einteilbar ist. Dann gilt sowohl $L_n \subseteq M$ als auch $L_n \subseteq M'$. Für M gilt dies, da in Punkt 4 alle einfachen Pfade, die von einem $v \in L_n$ zu einem Blatt führen, bei v als Typ-1-Pfade gezählt werden. Beim betreten der While-Schleife trifft dann für jeden dieser Knoten Bedingung 5.3 zu. Mit Beobachtung 3.3 können wir $L_n \subseteq M'$ annehmen. Im Weiteren nehmen wir an, dass r nicht in M ist. Sonst löschen wir r aus der pds, nehmen den einzigen Knoten in L_1 auf und erhalten eine äquivalente gleich große pds wie M.
Wir wissen bereits, dass $L_n \cap M = L_n \cap M'$ gilt, wenn n der Index der tiefsten Schicht ist. Wir wollen nun M' in eine pds M'' umformen, sodass für alle aufeinander folgenden Schichten $L_i \cap M = L_i \cap M''$ gilt, welches durch den folgenden Hilfsalgorithmus erfolgt:

1. $M'' = M'$
2. $i \leftarrow n - 1$
3. 1: **while** $i > 0$ **do**
   2: **if** $\exists\, v \in L_i \cap M''$ mit $v \notin L_i \cap M$ **then**
   3: z $\leftarrow parent_{V_H}(v)$
   4: **if** $z \in M''$ **then**
   5: M'' nicht optimal. **exit**.
   6: **else**
   7: $M'' = (M'' \setminus \{v\}) \cup \{z\}$
   8: **end if**
   9: **end if**
   10: **if** $\exists\, v \in L_i \cap M$ mit $v \notin L_i \cap M''$ **then**
   11: M'' ist keine pds. **exit**.
   12: **end if**
   13: $i \leftarrow i - 1$
   14: **end while**

Wir wollen nun zeigen, dass $L_b \cap M'' = L_b \cap M$ für $n \geq b > i$ eine Invariante des Hilfsalgorithmus ist. Für $i = n$ wissen wir dies bereits. Nehmen wir nun an für ein $i < n$ gilt selbiges. Erfüllen wir die Bedingung 3.2, aber 3.4 nicht, so ersetzen wir v in M'' durch $parent_{V_H}(v)$. Auf den Schichten L_n bis L_{i+1} sind M und M'' in diesem Moment identisch zueinander. In v münden dann bezüglich M bzw. M'' die gleichen Pfadtypen von unten. Da $v \notin M$ gilt, bedeutet dies, dass T_v über den Pfad P'' zwischen v und $parent_{V_H}(v)$ observierbar ist. Wenn nun in M'' der Knoten u durch $parent_{V_H}(v)$ ersetzt wird, so ist P'' observiert und somit auch T_v. Da am Knoten $parent_{V_H}(v)$ kein Pfad plötzlich unobserviert wird, bleibt dort alles gleich und M'' ist weiterhin eine pds. Danach wird die Invariante nicht mehr durch v verletzt. In 3.4 können wir ein PMU von einem Knoten $v \in M''$ nach $parent_{V_H}(v)$ verschieben und T_v ist immer noch observiert. Allerdings gilt $parent_{v_H}(v) \in M''$. Also hätten wir die PMU auf v auch löschen können, anstatt sie zu verschieben. Dies ist aber ein Widerspruch zur Minimalität von M''.
Erfüllen wir Bedingung 3.10, so kann man v durch $parent_{V_H}(v)$ in M ersetzen. Wieder sind M und M'' auf den Schichten L_n bis L_{i+1} identisch. Da gilt $v \notin M''$, muss es gereicht haben, den Pfad zwischen v und $parent_{V_H}(v)$ zu observieren, damit T_v observiert ist. Also liefert die Ersetzung von v durch $parent_{V_H}(v)$ in M wieder eine gültige pds. Dies ist aber ein Widerspruch zu Beobachtung 3.9. Also kann 3.10 laut Voraussetzung nie eintreten.
Für alle $v \in L_i$, die 3.2 nicht erfüllen, gilt entweder $v \in M$ und $v \in M''$ oder $v \notin M$ und $v \notin M''$. Also gilt nun ebenfalls $L_i \cap M'' = L_i \cap M$. Ist der Algorithmus beendet, folgert man deswegen $M'' = \dot{\bigcup}_{i=0\ldots n} L_i \cap M'' = \dot{\bigcup}_{i=0\ldots n} L_i \cap M = M$. Somit haben wir $|M'| = |M''| = |M| > |M'|$. Ein Widerspruch.

□

Proposition 3.11. *Algorithmus 1 besitzt eine Laufzeit von $O(|V|)$*

Beweis. Die Breitensuchvariante in Punkt zwei hat Laufzeit $O(|V| + |E|)$ und Punkt vier ist ebenfalls in dieser Zeit zu bewerkstelligen. In Punkt fünf und in der in jedem Durchlauf benutzten Prozedur follow taucht jeder Knoten und jede Kante nur einmal in der Bearbeitung auf. Dies benötigt nicht mehr als $O(|V| + |E|)$ Zeit. Der Rest braucht konstante Zeit. Da für einen Baum $|E| = |V| - 1$ gilt, folgt eine Gesamtlaufzeit von $O(|V|)$.

□

3.5 Vereinfachung des Algorithmus

Im bisher vorgestellten Algorithmus zur Lösung von PDS auf Bäumen, mussten wir die observierten und unobservierten Pfade während des Verfahrens unterscheiden. Je nachdem in welcher Kombination diese Pfade an einem Knoten aus V_H auftauchen, müssen wir entscheiden, ob der Knoten observierend sein soll. Wir können diesen Aufwand verringern und stellen deshalb einen zweiten Algorithmus vor, der PDS auf Bäumen löst. Dieser entscheidet ohne die Pfadtypen, wann ein Knoten observierend sein muss und wendet wenn möglich die Regel $\mathcal{G}$ an. Hierzu verwenden wir eine Art Marke für jeden Knoten v, die durch $obs(v)$ repräsentiert wird. Sie drückt aus, ob eine Knoten bereits observiert ist oder nicht.

Algorithm 2 *Berechnet eine kleinstmögliche pds für einen Baum*

Eingabe: Ein Baum $G(V, E)$.

Ausgabe: Eine Menge $P \subseteq V$, so dass jeder Knoten observiert und $|P|$ minimal ist.

1: $P \leftarrow \emptyset$.
2: Füge in L die Knoten geordnet nach ihrem absteigenden Abstand zur Wurzel ein.
3: **for all** $u \in L$ **do**
4: $\quad obs(u) \leftarrow f$.
5: **end for**
6: **while** $L \neq \emptyset$ **do**
7: $\quad v \leftarrow L.pop$
8: $\quad$ **if** $\exists v_1, v_2 \in child(v)$ mit $obs(v_1) \neq t$ und $obs(v_2) \neq t$ **then**
9: $\quad\quad obs(v) \leftarrow t$, $P \leftarrow P \cup \{v\}$, $obs(parent(v)) \leftarrow t$.
10: $\quad$ **else if** $obs(v) = t$ und $\forall v' \in child(v)$ gilt $obs(v') = t$ **then**
11: $\quad\quad obs(parent(v)) \leftarrow t$
12: $\quad$ **end if**
13: **end while**
14: **if** $obs(r) \neq t$ **then**
15: $\quad P \leftarrow P \cup \{r\}$
16: **end if**

Proposition 3.12. *Algorithmus 2 berechnet eine pds für einen Baum.*

Beweis. Im Laufe des Algorithmus gibt es für den aktuellen Knoten v genau drei Möglichkeiten bezüglich der Observation seiner Kinder:

1. v hat genau ein Kind u mit $obs(u) = f$.
2. v hat eine Teilmenge von Kinder $\{u_1, \ldots, u_k\}$, $k \geq 2$ mit $obs(u_i) = f$.
3. Für alle Kinder u von v gilt $obs(u) = t$.

Gilt Fall 1, dann können wir einen Knoten weiter oben im Baum in die pds nehmen, entweder den Vater von v oder einen, der noch näher an der Wurzel liegt. Denn sind der Knoten v, der Vater von v und alle Kinder von v außer u observiert, wenden wir $\mathcal{G}$ auf v an womit u observiert ist. Der Algorithmus muss hier also nichts tun und kann die Erweiterung der pds aufschieben.
In Fall 2 hat der Vaterknoten, falls er observierend ist, keinen Einfluss mehr auf die unobservierten Kinder $\{u_1, \ldots, u_k\}$. Deshalb muss der Knoten v in die pds genommen und der Vater kann als observiert markiert werden. Damit sind alle Knoten u unter v observiert. Der Teilbaum T_v hat nun keinen Einfluss mehr auf den Restgraphen und alle Marken der Knoten von T_v können implizit auf t gesetzt werden.
In Fall 3 sind alle Kinder von v observiert. Hier unterscheiden wir zwei Unterfälle:

1. Es gilt $obs(v) = f$. Dann gilt für alle Kinder von v, dass es genau ein Kind v' gibt mit $obs(v') = f$. Gäbe es ein Kind u mit Kindern u', u'' mit

$obs(u') = f$ und $obs(u'') = f$, dann würde $u \in S$ gelten und $obs(v) = t$ folgen. Gibt es ein Kind u von v, sodass für jedes Kind u' von u $obs(u') = t$ gilt, folgt wiederum durch Regel $\mathcal{G}$ $obs(v) = t$. Damit alle Knoten unter den Kindern von v observiert sind, reicht es dann v zu observieren. Dies kann wiederum durch den Vater von v geschehen. Also muss hier die pds noch nicht erweitert werden.

2. Es gilt $obs(v) = t$. Dann gilt für jedes Kind u entweder $obs(u') = t$ für alle Kinder u' von u oder es gibt maximal nur ein Kind u' von u mit $obs(u') = f$. Im letzteren Fall wird aber u' dadurch, dass $obs(v) = t$, $obs(u) = t$ und für alle anderen Kinder u'' von u $obs(u'') = t$ gilt, durch Regel $\mathcal{G}$ angewandt auf u observiert. Dann sind jedoch alle Knoten unter v observiert und ihre Marken sind implizit auf t gesetzt. Jetzt müssen wir nur noch Regel $\mathcal{G}$ auf v anwenden, weswegen wir dann $obs(parent(v)) \leftarrow t$ setzen.

Kommt der Algorithmus zur Wurzel r und gilt dann $obs(r) = f$, dann muss r in die pds aufgenommen werden, damit alle Vorfahren observiert sind. □

Algorithmus 2 berechnet also eine pds. Nun geht es darum, dass diese pds kleinstmöglich ist. Wir werden nun zeigen, dass die beiden Algorithmen für Bäume exakt die gleichen Mengen berechnen. Da wir von Algorithmus 1 wissen, dass er eine kleinstmögliche pds berechnet, folgt hieraus dieses ebenso für Algorithmus 2

Proposition 3.13. *Algorithmus 2 berechnet in $O(n)$ Zeit eine kleinstmögliche pds.*

Beweis. Sei M die Lösung, die Algorithmus 2 berechnet, und M', die Lösung die Algorithmus 1 berechnet. Gilt $r \in M'$, so ersetzen wir r durch den Knoten in L_1 und erhalten eine pds gleicher Größe. Wir wollen für alle Schichten L_i dann $L_i \cap M = L_i \cap M'$ zeigen. Klar ist auch, dass für alle $m \in M$ $\delta(m) > 2$ gilt. Denn es werden in Algorithmus 2 nur solche Knoten in die pds genommen, die mindestens zwei unobservierte Kinder haben, und Knoten mit $\delta(m) = 2$ haben nur ein Kind. Klar ist auch $L_n \subseteq M$, denn die Knoten in L_n sind die ersten, die mindestens zwei unobservierte Kinder haben. Wie wir bereits wissen gilt gleichfalls $L_n \subseteq M'$. Wir machen nun eine Induktion über die Schichten. Es gibt also ein i, sodass die Behauptung für L_j mit $n \geq j \geq i$ gilt. Zu zeigen ist nun, dass sie ebenfalls für L_{i-1} gilt. Sei nun $u \in L_i$. Es gibt zwei Möglichkeiten wie unsere Behauptung bezüglich u verletzt werden kann. Die erste trifft zu, wenn $parent_{V_H}(u) \in M'$ und $parent_{V_H}(u) \notin M$ gelten. Dann muss mit Beobachtung 3.9 aber $parent_{V_H}(u) \in M$ gelten, da sonst M keine pds wäre. Also Widerspruch. Die zweite Möglichkeit trifft zu, wenn $parent_{V_H}(u) \notin M'$ und $parent_{V_H}(u) \in M$ gelten. Dann muss $parent_{V_H}(u) =: s$ zwei Kinder u', u'' besitzen mit $obs(u') = f$ und $obs(u'') = f$ und vor dem Zeitpunkt als s zu M hinzugenommen wurde, galt $obs(s) = f$. Wenn wir u' und u'' jeweils denn Baum hinunter folgen, so können wir zwei Pfade $P_1 = s, u', q_1 \dots q_r$ und $P_2 = s, u'', q'_1 \dots q'_l$ finden, sodass für alle q_i und q'_i, mit $1 \leq i < r$ und $1 \leq j < l$, dann $obs(q_i) = f$ bzw. $obs(q'_i) = f$ gilt und q_r und q'_l $obs(q_r) = t$ bzw. $obs(q'_l) = t$ genügen. Da P_1 und P_2 mindestens drei Knoten besitzen, folgt mit Lemma 3.6.1, dass sie keine Typ-2-Pfade sein können. Da sie unobserviert sind, müssen sie dann aber Typ-1-Pfade sein. Deshalb folgt dann $parent_{V_H}(u) \in M'$. Ein Widerspruch zur

Annahme. Also gilt die Behauptung nun für die Schicht L_{i-1}. Also berechnet Algorithmus 2 eine kleinstmögliche pds.
Um die Liste L aufzustellen müssen wir wieder eine Breitensuche machen. Danach wird jeder Knoten zweimal betrachtet. Also ist die Laufzeit in $O(n)$. □

Für Bäume ergab sich letztlich ein recht einfacher Algorithmus. Es ist dennoch recht schwierig den nicht lokalen Charakter von PDS in den Griff zu bekommen. Der Grund hierfür ist die Regel $\mathcal{G}$. Wir werden im nächsten Kapitel erfahren, dass Kreise in einem Graphen zu großen Schwierigkeiten führen. In diesem Fall tritt der nicht lokale Charakter von PDS noch deutlicher zu Tage als bei Bäumen. Es wird ungleich schwerer aus lokalen Eigenschaften eines Knotens etwas über die globale Observation zu sagen.

Kapitel 4

Power Domination auf Generalisierten Series-Parallel Graphen

Im Folgenden wollen wir einen Linearzeit-Algorithmus zur Lösung von Power Dominating Set auf Generalisierten Series-Parallel Graphen entwickeln. Dazu verwenden wir die allgemeine Entwurfsmethode zur Entwicklung von Algorithmen für Series-Parallel Graphen aus K. Takamizawa et al. [20]. Als erstes wird die Klasse der Generalisierten Series-Parallel Graphen vorgestellt.

4.1 Generalisierte Series-Parallel Graphen

Aus graphentopologischer Sicht ist ein *Generalisierter Series-Parallel Graph* ein zusammenhängender Graph, der keinen Teilgraphen besitzt der homeomorph zum vollständigen Graphen auf vier Knoten K_4 ist. Zwei Graphen G_1, G_2 sind *homeomorph*, falls G_1 aus G_2 durch folgende zwei Operationen auf Knoten vom Grad zwei entsteht:

1. Sei $v \in V$ und $\delta(v) = 2$. Lösche v und verbinde seine beiden Nachbarn mit einer Kante.

2. Sei $\{a, b\} \in E$. Lösche $\{a, b\}$ aus E und nimm Kanten $\{a, v\}, \{v, b\}$ hinzu, wobei v ein neuer Knoten ist.

Die Klasse der Generalisierten Series-Parallel Graphen enthält die Klasse der Bäume und deswegen treten wir dadurch ein Stück näher an allgemeine Graphen heran. Ebenfalls ist jeder outerplanare Graph ein Generalisierter Series-Parallel Graph. Nach Wald und Colburn [21] sind die Partial-2-Trees genau die Graphen, die keinen Teilgraphen enthalten, der homeomorph zu K_4 ist. Folglich sind Generalisierte Series-Parallel Graphen genau die Partial 2-Trees. Die Partial-2-Trees sind aber (Bodlaender [6]) genau die Graphen mit Baumweite zwei. Folglich sind Generalisierte Series-Parallel Graphen ebenso genau die Graphen mit Baumweite zwei. Weiter ist nach R. J. Duffin [8] die Klasse der Generalisierten Series-Parallel Graphen äquivalent zu der Graphklasse, die

durch die folgende rekursive Definition entsteht. In dieser Definition besitzt jeder Generalisierte Series-Parallel Graph zwei ausgezeichnete Knoten u und v, die *Terminale* genannt werden. Die Notation hierfür ist $G(V, E, u, v)$.

4.1.1 Rekursive Definition

Definition 4.1. *Ein Generalisierter Series-Parallel Graph (GSP-Graph) ist rekursiv wie folgt definiert:*

1. Der vollständige Graph $K_2 = (\{u, v\}, \{\{u, v\}\})$ ist ein Generalisierter Series-Parallel Graph mit Terminalen u und v. Er ist der *Basisgraph* für die Klasse der Generalisierten Series-Parallel Graphen.

2. Sind zwei Generalisierte Series-Parallel (GSP-) Graphen $G_1(V_1, E_1, u_1, v_1)$ und $G_2(V_2, E_2, u_2, v_2)$ gegeben, dann ist der Graph G_3, der durch Anwendung einer der drei folgenden Regel auf G_1 und G_2 entsteht, ebenfalls ein Generalisierter Series-Parallel Graph:

   (a) *Series-1 Komposition (S1):* Identifiziere die Knoten v_1 und u_2, um $G_3(V_3, E_3, u_1, v_2)$ zu erhalten:
   $G_1(V_1, E_1, u_1, v_1) \odot_{s_1} G_2(V_2, E_2, u_2, v_2) \to G_3(V_3, E_3, u_1, v_2)$
   (siehe Abb. 4.1(a))

   (b) *Series-2 Komposition (S2):* Identifiziere die Knoten v_1 und u_2, um $G_3(V_3, E_3, u_1, v_1)$ (bzw. $G_3(V_3, E_3, u_1, u_2)$) zu erhalten:
   $G_1(V_1, E_1, u_1, v_1) \odot_{s_2} G_2(V_2, E_2, u_2, v_2) \to G_3(V_3, E_3, u_1, v_1)$
   (siehe Abb. 4.1(b))

   (c) *Parallele Komposition (P):* Identifiziere die Knoten u_1 und u_2, als auch v_1 und v_2, um $G_3(V_3, E_3, u_1, v_1)$ zu erhalten:
   $G_1(V_1, E_1, u_1, v_1) \odot_p G_2(V_2, E_2, u_2, v_2) \to G_3(V_3, E_3, u_1, v_1)$.
   Es wird angenommen, dass durch dieses Operation Mehrfachkanten nicht erzeugt werden. (siehe Abb. 4.1(c))

3. Nur Graphen, die nach einer endlichen Anzahl von Series-1, Series-2 und Parallelen Kompositionen erzeugt werden, sind Generalisierte Series-Parallel Graphen

In diesem Kapitel stellen u und v immer die beiden Terminale eines GSP-Graphen $G(V, E, u, v)$ da. Besitzen u und v zusätzlich noch einen Index, also u_i und v_i, so sind sie die Terminale des GSP-Graphen G_i.

Falls die Konstruktionssequenz eines Graphen nicht die S2-Komposition enthält, ist er ein *Series-Parallel Graph (SP-Graph)*. Bezeichnen wir durch $E(G)$ alle Kanten in einem Graphen G, so kann $E(G)$, falls G nach obiger Definition konstruiert wurde, in $E(G_1)$ und $E(G_2)$ partitioniert werden, wobei G_1, G_2 ebenfalls Generalisierte Series-Parallel Graphen sind.

4.1.2 Parse-Trees

Die Struktur jedes Generalisierten Series-Parallel Graphen G kann durch einen Parse-Tree beschrieben werden.

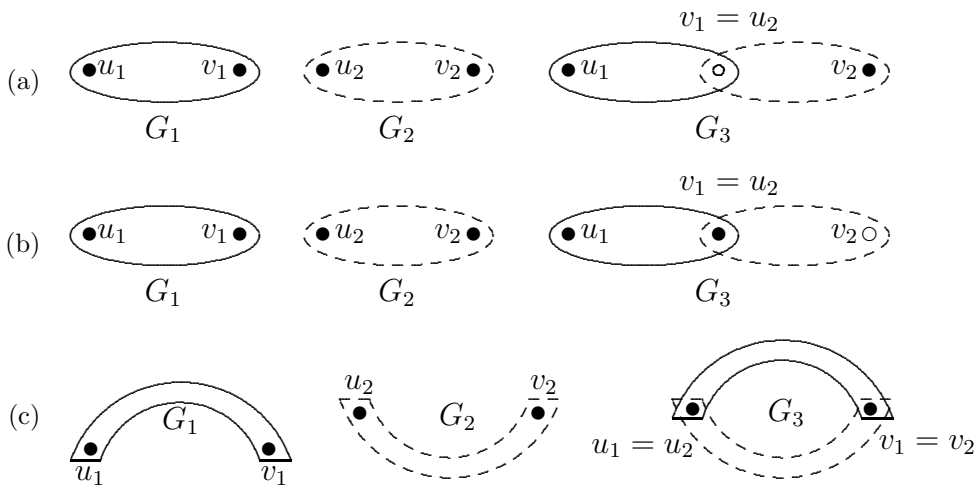


Abbildung 4.1: *(a) Series-1 Komposition, (b) Series-2 Komposition, (c) Parallele Komposition. Ausgefüllte Knoten kennzeichnen die Terminale.*

Definition 4.2. *Sei G ein GSP-Graph. Der* Parse-Tree *von G, bezeichnet durch $PT(G)$, ist ein Binärbaum in dem jeder Knoten einen Teilgraphen von G repräsentiert:*

1. *Jedes Blatt in $PT(G)$ steht für eine Kante in G. Es besteht eine 1-zu-1-Beziehung zwischen den Blättern von $PT(G)$ und den Kanten von G.*
2. *Jeder innere Knoten von $PT(G)$ hat ein Etikett $S1$, $S2$ oder P. Ein Knoten u mit dem Etikett $S1$ (bzw. $S2$ oder P) repräsentiert den Teilgraph von G, der entsteht, wenn eine Series-1 (bzw. Series-2 oder Parallele) Komposition auf die beiden Kinder von u angewendet wird.*
3. *Die Wurzel von $PT(G)$ repräsentiert G selbst.*

Es muss beachtet werden, dass ein GSP-Graph mehrere Parse-Trees haben kann. Abbildung 4.2 zeigt einen GSP-Graphen mit zwei Parse-Trees. Ein Parse-Tree $PT(G)$ gibt also eine Sequenz von Regeln S1, S2, und P auf E an, durch die G konstruiert werden kann. Es gibt einen Algorithmus in [20, 16] der einen Parse-Tree $PT(G)$ von G in linearer Zeit aufbauen kann. Mit Hilfe der Parse-Tree-Struktur werden wir einen effizienten Algorithmus konstruieren, um Power Dominating Set auf GSP-Graphen zu lösen.

4.2 Domination auf Generalisierten Series-Parallel Graphen

Der Algorithmus mit dem wir dieses Problem lösen wollen, stellt eine Art Dynamic Programming auf dem Parse-Tree des Graphen dar. Diese generelle Entwurfsmethode zur Entwicklung von Algorithmen für Series-Parallel Graphen wurde in [20] von K. Takamizawa et al. vorgestellt. T. Kikuno, N. Yoshida und Y. Kakuda verwenden diese Methode in [13] um einen Linearzeit-Algorithmus

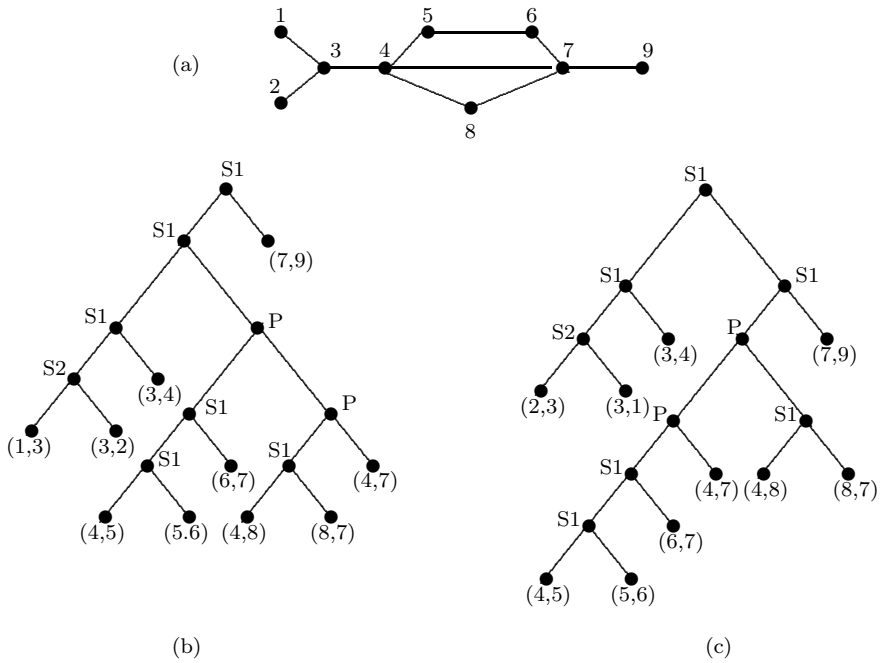


Abbildung 4.2: *(a) GSP-Graph G und zwei Parse-Trees (b) und (c) von G.*

für Dominating Set auf Series-Parallel Graphen zu entwickeln. Ausgehend von diesem Algorithmus wollen wir einen weiteren Algorithmus entwickeln, um Power Dominating Set zu lösen.

4.2.1 Ein Algorithmus für Dominating Set

Der Algorithmus aus [13] benutzt das Dynamic-Programming-Paradigma derart, dass für einen GSP-Graph G_3, der durch die Komposition von zwei GSP-Graphen G_1, G_2 entsteht, die optimale Lösung aus Teillösungen für G_1 und G_2 berechnet wird. Ein wichtiges Konzept hierbei ist der Begriff des Terminalzustandes. Sei $G(V, E, u, v)$ ein SP-Graph und $G'(V', E', u', v')$ ist ein Subgraph von G, der zu einem inneren Knoten von $PT(G)$ korrespondiert. Nehmen wir nun an, wir haben bereits eine kleinstmögliche Dominating Set D für G. Dann unterscheiden wir drei Arten, wie ein Terminal u' bzw. v' in G' dominiert wird:

1. Zustand I: $u' \in D$.
2. Zustand II: Es gibt $d \in D \cap V'$, welches das Terminal u' dominiert.
3. Zustand III: Es gibt $d \in D \cap (V \setminus V')$, welches u' dominiert.

In einem Initialisierungsschritt wird eine Kante unter der Prämisse betrachtet, dass an den Terminalen einer der drei Zustände herrscht. Für jede dieser neun Kombinationen aus Terminalzuständen wird eine kleinstmögliche Dominating Set bestimmt. Dann wird der Parse-Tree $PT(G)$ auf Bottom-Up-Weise traversiert. Für jeden Graphen zu einem inneren Knoten werden wieder neun kleinstmögliche Lösungen, jeweils eine für alle Kombinationen aus Terminalzuständen, aus den Lösungen der Kinder berechnet. Sind wir dann an der Wurzel

von $PT(G)$ angelangt, so können wir die Lösung für G bestimmen. Wir betrachten nun alle Lösungen für G, sodass an den Terminalen einer der Zustände I oder II herrscht. Die Möglichkeiten in denen ein Terminal Zustand III besitzt, sind nicht relevant. Denn in diesem Fall ist das Terminal nicht dominiert. Die kleinstmögliche Lösung hieraus ist die kleinstmögliche Dominating Set für G.

4.2.2 Domination versus Power Domination

Eine wichtige Eigenschaft einer Dominating Set ist ihre Lokalität. Für jeden Knoten wissen wir, dass entweder er selbst oder einer seiner Nachbarn in einer kleinstmöglichen Dominating Set enthalten sein muss. Eine Power Dominating Set besitzt diese Eigenschaft aufgrund von Regel $\mathcal{G}$ nicht. Es ist möglich, dass ein Knoten observiert ist, obwohl weder er selbst noch einer seiner Nachbarn in der Power Dominating Set enthalten ist. Versuchen wir nun den oben angesprochenen Algorithmus für Dominating Set auf Power Dominating Set zu übertragen, so wird schnell klar, dass die drei Terminalzustände nicht ausreichen. In keinem der drei Terminalzustände findet Regel $\mathcal{G}$ ihre Entsprechung. Eine Idee, die sich sofort aufdrängt, ist, dass wir einfach weitere Terminalzustände einführen. Diese müssen ausdrücken, dass ein Terminalknoten u' durch Regel $\mathcal{G}$, angewendet auf einen Nachbarknoten, observiert werden kann. Dieser Nachbarknoten kann sowohl innerhalb als auch außerhalb des Teil-Series-Parallel Graphen liegen, in dem u' Terminal ist. Sei also D eine Power Dominating Set, so ergänzen wir die Terminalzustände aus Abschnitt 4.2.1 um zwei weitere:

4. Zustand IV: Es gibt $w \in V'$, so dass u' mit Regel $\mathcal{G}$ angewendet auf w observiert wird.

5. Zustand V: Es gibt $w \in V \setminus V'$, so dass u' mit Regel $\mathcal{G}$ angewendet auf w observiert wird.

Diese Ergänzungen treffen aber leider nicht den Kern des Problems. Betrachten wir die beiden Series-Parallel Graphen in Abbildung 4.3.

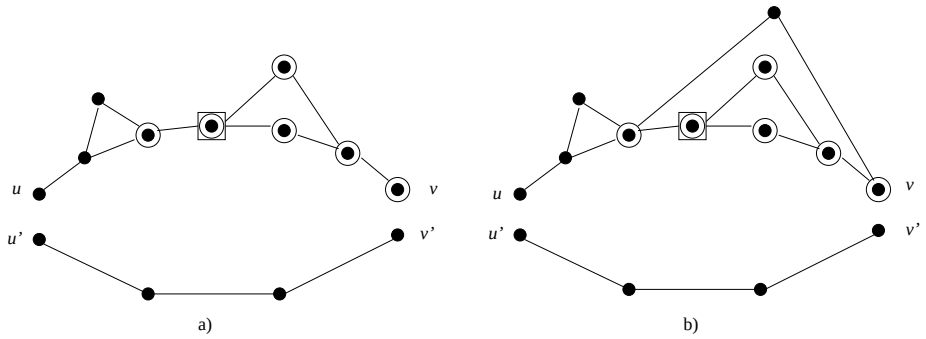


Abbildung 4.3: *In a) und b) sind jeweils zwei Series-Parallel Graphen zu sehen. Die umrandeten Knoten sind observiert.*

In Abbildung 4.3 a) sind zwei Series-Parallel Graphen zu sehen. Im oberen Graph sind schon teilweise Knoten observiert. Wir wollen nun sehen, was bezüglich Observation geschieht, wenn wir u mit u' und v mit v' identifizieren.

Der Nachbar von $v = v'$ im unteren Graphen wird sofort mit Regel $\mathcal{G}$ observiert. Observation pflanzt sich mittels Regel $\mathcal{G}$ über den ganzen unteren Graphen fort und am Ende ist sogar der ganze obere Graph observiert.
Wenn wir in b) die gleichen Knotenidentifikationen vornehmen, ist das Verhalten bezüglich Observation aber vollkommen verschieden. Da $v = v'$ noch einen zusätzlichen unobservierten Knoten im oberen Graph besitzt, kann Regel $\mathcal{G}$ nicht auf $v = v'$ angewendet werden. Somit bleibt der Nachbar von $u = u'$ im unteren Graph unobserviert. Insgesamt wird durch die P-Komposition kein weiterer Knoten observiert. Sowohl in a) als auch in b) können wir dem Terminal v den Zustand V zuordnen, da es von innerhalb mit $\mathcal{G}$ observiert wird. Damit die Zustände ausreichen, müssen sie einen gewissen Grad an Abstraktion bieten. Es muss also nur anhand des Zustands entscheidbar sein, wie sich eine Komposition zweier GSP-Graphen auf deren Observation auswirkt. Aber im oberen Fall haben wir verschiedenes Observationsverhalten beobachten können, obwohl an v der gleiche Zustand sowohl in a) als auch in b) herrschte und der untere Graph beide Male derselbe war. Der letztliche Grund für das unterschiedliche Verhalten in b) ist ein zusätzlicher unobservierter Knoten innerhalb des oberen Graphen. Es handelte sich also um einen Teil der inneren Struktur des Graphen. Dies ist der Hauptunterschied von Power Dominating Set zu Dominating Set. Bei DS spielt die innere Struktur keine Rolle. Denn zwei Graphen, die miteinander komponiert werden, können sich gegenseitig nur an den Terminalen beeinflussen. Power Dominating Set ist da viel komplexer. Hier werden auch Knoten innerhalb der beiden Kompositionskomponenten mittels Regel $\mathcal{G}$ beeinflusst.

4.3 Gültige Ausrichtungen

In diesem Abschnitt werden wir ein neues graphentheoretisches Konstrukt, die Gültige Ausrichtung, kennen lernen. Gültige Ausrichtung und Power Dominating Set stehen in enger Verwandtschaft, wie wir im Weiteren sehen werden. Dieses neue Konstrukt wird uns helfen zu entscheiden, welche Information über die innere Struktur eines Graphen relevant zur Berechnung einer Power Dominating Set ist. Insbesondere können wir Power Domination bezüglich der Regel $\mathcal{G}$ feingranularer betrachten. Es ist nämlich möglich, dass ein Graph durch eine Knotenmenge observiert wird, dies aber durch unterschiedliche Abfolgen von Regel-$\mathcal{G}$-Anwendungen geschieht. Betrachten wir hierzu Abbildung 4.4. Wir se-

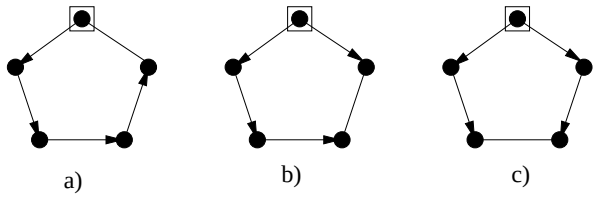


Abbildung 4.4: *Ein Graph und 3 Möglichkeiten ihn durch verschiedene Regel-$\mathcal{G}$-Anwendungen zu observieren. Eine gerichtete Kante (u, v) bedeutet, dass v durch Regel $\mathcal{G}$ auf u angewendet observiert wird, falls u nicht observierend ist. Falls u observierend ist, bedeutet es, dass v durch Regel $\mathcal{L}$ auf u angewendet, observiert wird.*

hen die durch gerichtete Kanten kenntlich gemachten Anwendungen der globalen und lokalen Regeln. In Bezug auf Power Domination können wir diese drei Objekte nicht unterscheiden. Innerhalb des Konzepts der Gültigen Ausrichtung sind diese aber klar voneinander verschieden. In anderen Worten ausgedrückt: Power Domination sagt uns nur, ob ein Graph observiert ist oder nicht. Die Gültige Ausrichtung enthält darüber hinaus auch noch Informationen, wie ein Graph observiert wird.

4.3.1 Eine neue Beschreibung von Power Domination

Wir wollen nun in das Konzept der Gültigen Ausrichtung einführen. Dies werden wir schrittweise über mehrere Definitionen tun.

Ausrichtungen

Eine Ausrichtung entsteht einfach dadurch, dass man in einem bestehenden Graphen die Kanten folgendermaßen ausrichtet.

Definition 4.3. *Sei $G(V, E)$ ein Graph. Eine Ausrichtung $R(V, F)$ für G ist ein gerichteter Graph, so dass für alle $e = \{u, v\} \in E$ entweder $(u, v) \in F$ und $(v, u) \notin F$ oder $(u, v), (v, u) \in F$ gilt und es für alle $(u, v) \in F$ eine Kante $\{u, v\} \in E$ gibt. Wir partitionieren F in zwei Mengen F_u und F_b, so dass gilt: $F_u := \{(u, v) \mid (u, v) \in F, (v, u) \notin F\}$ $F_b := \{(u, v) \mid (u, v), (v, u) \in F\}$.*

Gilt für eine Kante $\{u, v\} \in E$, dass (u, v) in F_u ist, so gibt es keine weitere Kante $(v, u) \in F$. Die Kante $\{u, v\}$ wurde also nur einmal ausgerichtet (unidirected). Gilt aber $(u, v) \in F_b$, so gibt es auch $(v, u) \in F$, $\{u, v\}$ ist also in beide Richtungen ausgerichtet worden (bidirected). Figurativ bedeutet in einer Ausrichtung eine Kante $(u, v) \in F_u$, dass v durch die Anwendung von Regel $\mathcal{G}$ auf u observiert wird, falls u nicht observierend ist. Sonst ist die Bedeutung, dass Observation durch eine Anwendung von $\mathcal{L}$ erfolgt. Ein Pfad aus solchen Kanten stellt dann ein Abfolge von Regel-$\mathcal{G}$-Anwendungen dar. Dies führt zur nächsten Definition.

Definition 4.4. *Ein gerichteter Pfad $P = (v_1, \ldots, v_k)$ in einer Ausrichtung $R(V, F)$ ist ein Observationspfad, falls für alle $i \in \{1, \ldots, k-1\}$ gilt $(v_i, v_{i+1}) \in F_u$.*

Betrachten wir nun einen Observationspfad $P = (v_1, \ldots, v_k)$. Dann steht der Observationspfad P dafür, dass v_k mit Regel $\mathcal{G}$ 'transitiv' angewendet auf v_1 observiert wird. Dies heißt für alle $i \in \{1, \ldots, k-1\}$ wird v_{i+1} mit Regel $\mathcal{G}$ angewendet auf v_i observiert. Ein Knoten v auf dem Observationspfad ist bezüglich Observation also von seinen Vorgängern abhängig.
Die Kanten in F_u werden im Weiteren von besonderer Wichtigkeit sein. Deswegen definieren wir:

Definition 4.5. *Sei $R(V, F)$ eine Ausrichtung und $A \subseteq V$. $R(V, F)$ ist eine Ausrichtung bezüglich A wenn folgende Punkte erfüllt sind:*

1. *Für alle $v \in V$ gilt: $indeg_{F_u}(v) = 0 \iff v \in A$.*
2. *Für alle $v \in V \setminus A$ gilt $indeg_{F_u}(v) = 1$ und $outdeg_{F_u}(v) \leq 1$.*
3. *Für alle $v \in V \setminus A$ gibt es einen Observationspfad $P = (a, w_1, \ldots, w_q, v)$ mit $a \in A$ und $w_1, \ldots, w_q \notin A$.*

Wir nennen A Bezugsmenge.

Eine beispielhafte Ausrichtung bezüglich einer Knotenmenge sehen wir in den Abbildungen 4.5 und 4.6 b), c). Wenn wir die Observationspfade als die transitive Anwendung der Regel $\mathcal{G}$ verstehen, so sind die Startknoten quasi die Quellen der Observation. Dies sollen hier genau die Knoten in der Bezugsmenge A sein, welches in Definition 4.5 gerade dadurch festgelegt wird, dass es keine Kanten (v, a) mit $a \in A$ gibt. Schlagen wir nun einen Bogen zu Power Domination, so wäre dort A die Menge der observierenden Knoten. Wenn wir verlangen, dass außerdem jeder Knoten auf einem Observationspfad liegt, so heißt dies für Power Domination, dass jeder Knoten observiert sein soll. Da jeder Knoten nur einmal mittels Regel $\mathcal{G}$ observiert werden kann, wollen wir, dass jeder Knoten aus $V \setminus A$ nur auf einem Observationspfad liegt. Dies erreichen wir mit der Beschränkung der inzidenten Kanten aus F_u, wie wir in folgender Beobachtung sehen werden.

Beobachtung 4.6. *Für zwei Observationspfade P_i, P_j, $P_i \neq P_j$, in einer Ausrichtung $R(V, F)$ bezüglich $A \subseteq V$ gilt folgendes*

1. *P_i und P_j haben höchstens einen Knoten gemeinsam.*
2. *Aus $v \in P_i$ und $v \in P_j$ folgt $v \in A$.*
3. *Es gibt keinen gerichteten Kreis, der nur aus Kanten aus F_u besteht.*

Beweis. Betrachten wir einen Knoten u mit $u \in P_i$ und $u \in P_j$. Dann kann nur entweder $indeg_{F_u}(u) > 1$ oder $outdeg_{F_u}(u) > 1$ oder beides gelten. In einer Ausrichtung bezüglich A kann es laut Definition 4.5 keinen Knoten mit $indeg_{F_u}(u) > 1$ geben. Gilt $outdeg_{F_u}(u) > 1$, so folgt $u \in A$. Für jeden weiteren Knoten $\bar{u}$ mit $\bar{u} \in P_i$ und $\bar{u} \in P_j$ folgt wiederum $\bar{u} \in A$. Da jeder Observationspfad aber nur einen Knoten aus A enthält, gilt $u = \bar{u}$. Somit sind 1 und 2 bewiesen.
Liegt u auf einem gerichteten Kreis, der nur aus Kanten aus F_u besteht, so gilt $indeg_{F_u}(u) = 1$ und $outdeg_{F_u}(u) = 1$. Dann kann u aber auf keinem Observationspfad mehr liegen. Widerspruch. □

Wenn nun die Kanten aus F_u als Regel-$\mathcal{G}$-Anwendungen zu verstehen sind, die Bezugsmenge A die observierenden Knoten darstellen soll und jeder Knoten auf einem Observationspfad liegt, dann könnte man doch annehmen, dass A eine Power Dominating Set ist. Versuchen wir allerdings in Abbildung 4.6 bezüglich der Ausrichtungen b) und c) die Menge der observierten Knoten zu berechnen, so sehen wir, dass nur in c) alle Knoten observiert werden. Wir sind also noch nicht ganz am Ziel angelangt.

Gültige Ausrichtungen

Die oben angesprochene Problematik lässt sich folgendermaßen erklären. Wenn eine Kante $(u, v) \in F_u$ eine Regel-$\mathcal{G}$-Anwendung darstellt, so kann diese Regel nur angewendet werden, wenn alle Nachbarn von u außer v bereits observiert sind. Betrachte hierzu Abbildung 4.5.

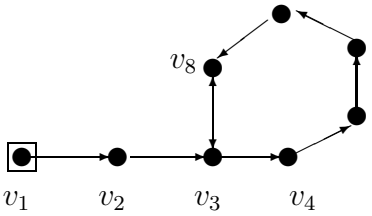


Abbildung 4.5: *Eine Ausrichtung bezüglich* $\{v_1\}$.

Hier kann v_4 nur durch $\mathcal{G}$ observiert werden, wenn zuvor auch v_3, v_2 und v_8 observiert sind. Der Knoten v_8 liegt auf dem gleichen Observationspfad wie v_4. Und da v_4 vor v_8 im Observationspfad liegt, ist v_8 nur observiert, wenn v_4 observiert ist. Wir sehen sofort, dass die Abhängigkeit zwischen v_4 und v_8 nicht auflösbar ist. Dies ist auch nicht verwunderlich, wenn wir die Konstellation im Hinblick auf Power Domination betrachten. Es ist ganz klar, dass die Bezugsmenge $\{v_1\}$ keine Power Dominating Set ist. Aufgrund dieses Sachverhalts brauchen wir noch folgende Begrifflichkeiten.

Definition 4.7. *Ein alternierender Kreis in einer Ausrichtung $R(V, F)$ ist ein gerichteter Kreis in dem keine zwei Kanten aus F_b inzident sind und kein Knoten aus der Bezugsmenge ist.*

Definition 4.8. *Eine Ausrichtung $R(V, F)$ bezüglich $A \subseteq V$ ist eine Gültige Ausrichtung (GA) bezüglich A, falls sie keinen alternierenden Kreis enthält.*

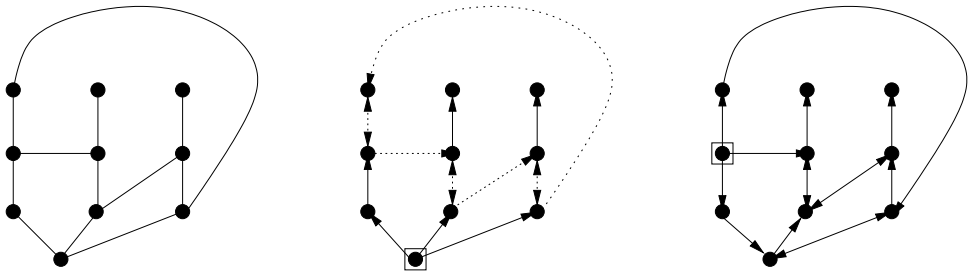


Abbildung 4.6: *In a) ist ein Graph $G(V, E)$ gegeben. In b) wird eine Ausrichtung für G gezeigt, die keine Gültige Ausrichtung ist. Der alternierende Kreis ist durch gepunktete Kanten deutlich gemacht. Bild c) zeigt eine Gültige Ausrichtung für G.*

Zum besseren Verständnis betrachten wir Abbildung 4.6. Dort sehen wir zu einem Graphen zwei Ausrichtungen. Die erste ist keine Gültige Ausrichtung,

da sie einen alternierenden Kreis besitzt, und die zweite ist eine Gültige Ausrichtung. Ein alternierender Kreis stellt also eine gegenseitige, nicht auflösbare Abhängigkeit der Knoten in Bezug auf $\mathcal{G}$ dar.

4.3.2 Äquivalenz von Power Dominating Set und Gültiger Ausrichtung

Jetzt wollen wir die Gleichartigkeit von Power Domination und Gültiger Ausrichtung beweisen. Dies soll bedeuten, dass es eine Power Dominating Set P für einen Graphen genau dann gibt, wenn eine Gültige Ausrichtung bezüglich P existiert.

Proposition 4.9. *Sei $G(V, E)$ ein Graph und $P \subseteq V$.*
P ist eine Power Dominating Set $\Rightarrow$ Es existiert eine Gültige Ausrichtung $R(V, F)$ bezüglich P für G.

Beweis. Wir wenden als erstes folgendes Verfahren auf G an:

1. $OV = P$, $F_u = \emptyset$.

2. Solange es $v \in N(p)$ mit $p \in P$, $v \notin OV$ gibt, setze $F_u = F_u \cup \{(p, v)\}$, $OV = OV \cup \{v\}$.

3. Solange es $v \in OV$, $u \in V \setminus OV$ mit $u \in N(v)$ und $N[v] \setminus \{u\} \subseteq OV$ gibt, setze $F_u = F_u \cup \{(v, u)\}$, $OV = OV \cup \{u\}$.

4. $F_b = \{(a, b), (b, a) \mid \{a, b\} \in E, (a, b) \notin F_u \text{ oder } (b, a) \notin F_u\}$.

Wir wollen jetzt im Folgenden zeigen, dass $R(V, F_u \cup F_b)$ eine Gültige Ausrichtung bezüglich P für G ist. Dass $R(V, F_u \cup F_b)$ eine Ausrichtung ist, ist klar. Wir beweisen nun, dass R auch eine Ausrichtung bezüglich P ist, indem wir die Punkte aus Definition 4.5 nachweisen. Bezüglich des Verfahrens sei Folgendes bemerkt: Unter Punkt 1 und 2 wird nichts anderes getan als Regel $\mathcal{L}$ anzuwenden, und Regel $\mathcal{G}$ findet ihre Entsprechung in Punkt 3.

1. Es werden Kanten $\{u, v\}$ immer nur ausgerichtet, also (u, v) zu F_u hinzugenommen, wenn $u \in OV$ und $v \notin OV$ gilt. Da von Anfang an $P \subseteq OV$ gilt, wird nie eine Kante so ausgerichtet, dass sie auf ein $p \in P$ zeigt. Also folgt: $v \in P \Rightarrow indeg_{F_u}(v) = 0$.
   Da P eine pds ist und das Verfahren nur die Anwendung von $\mathcal{L}$ und $\mathcal{G}$ darstellt, muss am Ende des Verfahrens jeder Knoten $v \in V$ auch in OV enthalten sein. Ein Knoten $v \in V \setminus P$ wird entweder in Punkt 2 oder 3 nach OV übernommen. Gleichzeitig wird aber auch eine Kante (u, v) nach F_u übernommen. Also gilt $indeg_{F_u}(v) \geq 1$. Folglich muss gelten: $indeg_{F_u}(v) = 0 \Rightarrow v \in P$.

2. Wir wissen bereits, dass $indeg_{F_u}(v) \geq 1$ für alle $v \in V \setminus P$ gilt. Im selben Moment wie die erste Kante (u, v) mit v als Endknoten nach F_u übernommen wird, wird v nach OV übernommen. Für alle Knoten $\tilde{v} \in OV$ gilt aber, dass nie wieder eine Kante $(\tilde{u}, \tilde{v})$ mit $\tilde{v}$ als Endknoten nach F_u übernommen wird. Also gilt $indeg_{F_u}(v) = 1$.
   Betrachte $v \in V \setminus P$ zum ersten Zeitpunkt wenn eine Kante $\{v, u\}$ als gerichtete Kante (v, u) nach F_u übernommen wird. Für alle $q \in N[v] \setminus \{u\}$

gilt $q \in OV$. Also muss es bereits eine Kante $(l, q) \in F_u$ mit $l \neq v$ geben. Da $indeg_{F_u}(q) = 1$ gilt, wird es keine Kante $(v, q) \in F_u$ geben. Folglich $outdeg_{F_u}(v) \leq 1$.

3. Da für alle $v \in V \setminus P$, $indeg_{F_u}(v) = 1$ gilt, kann man von v aus die Kanten aus F_u in entgegengesetzter Richtung eindeutig verfolgen. Treffen wir auf diese Weise auf einen Knoten, den wir schon einmal betrachtet haben, so kann dieser nicht in der Bezugsmenge P enthalten sein und wir haben einen Kreis gefunden, der nur aus Kanten aus F_u besteht. Dies kann aber nicht sein, da wir sonst eine Kante (v, u) nach F_u übernommen haben, obwohl $u \in OV$ gilt. Also müssen wir auf einen Knoten a mit $indeg_{F_u}(a) = 0$ treffen. Dann gilt $a \in P$ und wir haben einen Observationspfad zwischen a und v gefunden.

Jetzt wissen wir, dass $R(V, F_u \cup F_b)$ eine Ausrichtung bezüglich P ist. Damit R eine Gültige Ausrichtung bezüglich P ist, müssen wir noch nachweisen, dass es keinen alternierenden Kreis in R gibt. Um dies zu bewerkstelligen nehmen wir eine Beobachtung vor, die wir im Weiteren benutzen wollen. Im obigen Verfahren fügen wir der Menge OV schrittweise weitere Knoten aus $V \setminus OV$ hinzu. Wir nummerieren nun die Schritte, in denen Knoten nach OV kommen. Wir führen hierzu Variablen t_v für alle $v \in V$ ein. Die Variable t_v gibt nun an, in welchem Schritt ein Knoten nach OV übernommen wurde.

Beobachtung 4.10. *Sei $v \notin P$*

1. *Für alle $(v, u) \in F_u$ gilt $t_v < t_u$*
2. *Für alle $(v, u) \in F_u$ und $(v, z), (z, v) \in F_b$ gilt $t_z < t_u$*

Beweis der Beobachtung:. Der Knoten u wurde in Punkt 3 des Verfahrens nach OV dadurch übernommen, dass zuvor $N[v] \setminus \{u\} \subseteq OV$ galt. Insbesondere galt zu diesem Zeitpunkt bereits $v, z \in OV$ und deshalb folgt $t_v < t_u$ und $t_z < t_u$. □

Um einen Widerspruch zu provozieren nehmen wir nun an, $R(V, F_b \cup F_u)$ besitzt einen alternierenden Kreis $C = v_1, \ldots, v_l$. Der alternierende Kreis C kann als eine Aufeinanderfolge von Abschnitten aus Observationspfaden P_i, welche immer durch exakt eine Kante aus F_b getrennt sind, aufgefasst werden, siehe Abbildung 4.7. Also gilt $C = P_1, \ldots, P_s$. Für alle $P_i = (v_{i_1}, \ldots, v_{i_{n_i}})$ können wir mit Beobachtung 4.10 $t_{v_{i_1}} < \ldots < t_{v_{i_{n_i}}}$ schließen. Für zwei aufeinander folgende Observationspfade P_i, P_{i+1} schließen wir mit Beobachtung 4.10, wenn $v_{i_{n_i}}$ der Endknoten von P_i und v_{i+1_2} der zweite Knoten von P_{i+1} ist, dass $t_{v_{i_{n_i}}} < t_{v_{i+1_2}}$ gelten muss. Ist dann aber v_1 o.B.d.A. kein Anfangsknoten von P_1, so folgt hieraus $t_{v_1} < t_{v_l}$ obwohl $v_1 = v_l$ gilt. Ein Widerspruch, $R(V, F_b \cup F_u)$ enthält doch keinen alternierenden Kreis und ist folglich eine Gültige Ausrichtung bezüglich P. □

Nun beweisen wir die Umkehrung der letzten Proposition.

Proposition 4.11. *Sei $G(V, E)$ ein Graph.*
$R(V, F)$ ist bezüglich $A \subseteq V$ eine Gültige Ausrichtung für $G \Rightarrow A$ ist eine Power Dominating Set für G.

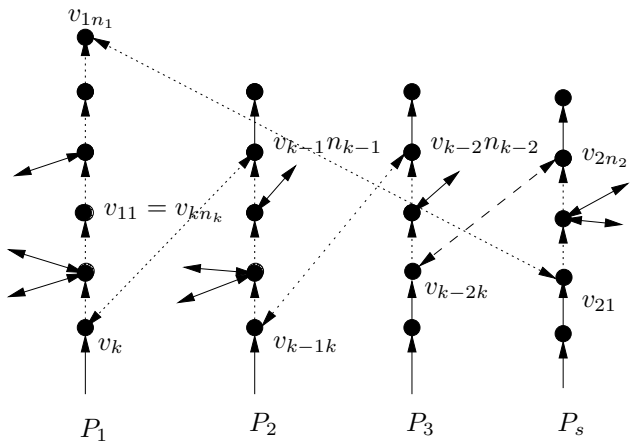


Abbildung 4.7: *Ein alternierender Kreis ist eine Abfolge von Observationspfaden.*

Beweis. Um diese Proposition zu beweisen brauchen wir ein Verfahren, welches dem aus Proposition 4.9 ähnelt.

1. $OV = N_G[A]$

2. Solange es $u \in V \setminus OV$ gibt mit $(v, u) \in F_u$ und $N_G[v] \setminus \{u\} \subseteq OV$, dann $OV = OV \cup \{u\}$.

Dieses Verfahren entspricht in Punkt 1 der Regel $\mathfrak{L}$. Der Punkt 2 ist prinzipiell, bis auf eine kleine Einschränkung, die Anwendung von Regel $\mathfrak{G}$. Damit ein Knoten u in Punkt 2 nach OV hinzugenommen wird, muss für einen Nachbarn v erstens $N[v] \setminus \{u\} \subseteq OV$ und zweitens zusätzlich noch $(v, u) \in F_u$ gelten. Aus diesem Grund ist klar, dass wenn das Verfahren beendet ist und $OV = V$ gilt, A eine Power Dominating Set ist.
Wir verwenden wiederum einen Widerspruchsbeweis. Nehmen wir an, das Verfahren hält an und es gilt $OV \neq V$. Dann gilt für einen Knoten $u \in OV$ entweder $N[u] \subseteq OV$ oder $|N[u] \cap (V \setminus OV)| > 1$. Unmöglich hingegen ist der Fall $|N[u] \cap (V \setminus OV)| = 1$, da sonst das Verfahren einen weiteren Schritt in Punkt 2 unternehmen kann. Sei $u_1 \in OV$ ein Knoten, für den Letzteres gilt. Seien $v_1, z_1 \in N[u_1] \cap (V \setminus OV)$. Da Punkt 2 auf u_1 nicht anwendbar ist, können wir $(u_1, v_1) \in F_u$ annehmen, siehe Abbildung 4.8.
Seien $P_1, \ldots, P_l$ alle Observationspfade in $R(V, F)$. Gelte dann o.B.d.A. $u_1, v_1 \in P_1$ und $z_1 \in P_2$. Folgen wir von z_1 aus P_2 in entgegengesetzter Richtung, so stoßen wir, da $z_1 \notin OV$, auf ein $u_2 \in OV$, so dass $|N[u_2] \cap (V \setminus OV)| > 1$ gilt. $|N[u_2] \cap (V \setminus OV)| = 1$ kann wieder nicht sein, da sonst das Verfahren in Punkt 2 einen weiteren Schritt unternehmen kann. Dann gibt es wieder Knoten $v_2, z_2 \in N[u_2] \cap (V \setminus OV)$ mit $(u_2, v_2) \in F_u$ und o.B.d.A. $z_2 \in P_3$.
Die gleiche Konstellation haben wir schon in Bezug auf Knoten u_1 beobachtet. Wir setzen also dieses Prozedere fort. Da der Graph R endlich ist, müssen wir irgendwann auf einen Observationspfad P_s stoßen, den wir bereits betrachtet haben. Genauer gesagt befinden wir uns in folgender Situation: $u_j \in OV$, $u_j \in P_j$ mit $|N[u_j] \cap (V \setminus OV)| > 1$, $v_j, z_j \in N[u_j] \cap (V \setminus OV)$, $(u_j, v_j) \in F_u$, $z_j \in P_s$ und es gilt $s \leq j$.

Die Knoten v_k und z_{k-1} liegen immer auf dem Observationspfad P_k.
Mit $v_k \to z_{k-1}$ meinen wir nun die Knoten, die zwischen diesen beiden Knoten auf P_k liegen.
Jetzt haben wir aber in der obigen Situation einen alternierenden Kreis C gefunden:

$$C = (z_j, u_j, v_j \to z_{j-1}, u_{j-1}, v_{j-1} \to \ldots \to z_{s+1}, u_{s+1}, v_{s+1} \to z_s, u_s, v_s \to z_j)$$

Abbildung 4.8 verdeutlicht diesen Sachverhalt an einem Beispiel. Die Existenz von C steht aber im Widerspruch dazu, dass $R(V, F)$ eine Gültige Ausrichtung bezüglich A ist. Folglich ist A dann eine pds. □

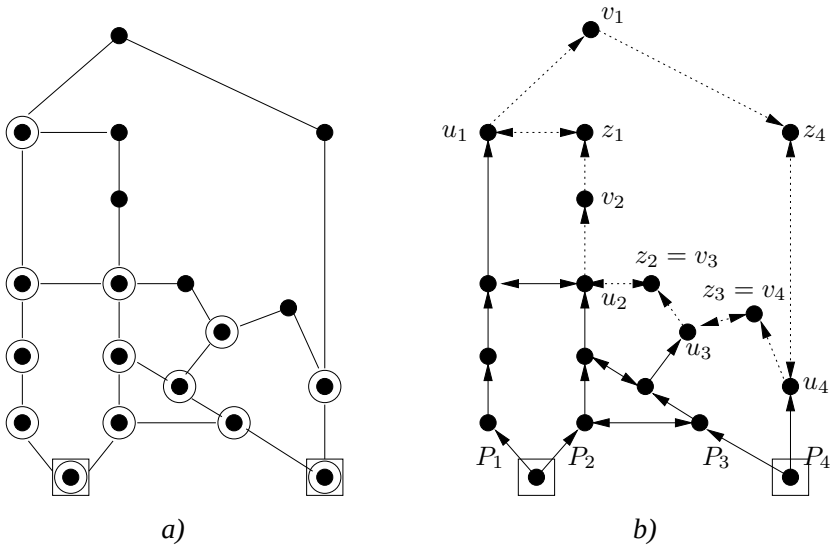


Abbildung 4.8: *In a) sind die Knoten in $OV \neq V$ nach Beendigung des Verfahrens und in b) der alternierende Kreis, der dann zu finden ist, zu sehen.*

Mit den beiden Propositionen 4.9 und 4.11 folgern wir die nächste Aussage.

Theorem 4.12. *Sei $G(V, E)$ ein Graph und $P \subseteq V$ dann gilt*
P ist eine Power Dominating Set $\iff$ Es gibt eine Gültige Ausrichtung $R(V, F)$ bezüglich P für G.

Hieraus schließen wir, dass die Ermittlung einer Power Dominating Set für einen Graphen $G(V, E)$ äquivalent dazu ist, eine Gültige Ausrichtung bezüglich einer kleinstmöglichen Bezugsmenge $A \subseteq V$ zu finden. Ein entscheidender Vorteil hierbei ergibt sich, wenn man einen Dynamic-Programming-Ansatz verfolgt. Denn hat man eine Lösung für ein Teilproblem, so ist es wichtig zu wissen, auf welche Art man sie in einer Lösung für das Gesamtproblem verwenden kann. Im Falle der Gültigen Ausrichtung ist es sofort ersichtlich wie man eine Teillösung verwendet. Zwei Teillösungen auf verschiedenen Subgraphen von G können miteinander kombiniert werden, wenn dadurch die Anforderungen an eine Gültige Ausrichtung nicht verletzt werden. Es darf insbesonders kein alternierender Kreis entstehen.
Wir vertreten außerdem den Standpunkt, dass zu einer Power Dominating Set

auch die Art und Weise der Regel-$\mathcal{G}$-Anwendung gehören. Also quasi die Angabe wer von wem observiert wird. Wenn wir dann bezüglich einer Power Dominating Set P alle möglichen Gültigen Ausrichtungen bezüglich P angeben, so ist P vollkommen charakterisiert. Denn eine Gültige Ausrichtung macht die Abhängigkeiten bezüglich Observation sichtbar.

4.4 Ein Linearzeit-Algorithmus

Um Power Dominating Set für GSP-Graphen zu lösen, betrachten wir ab hier nur noch das äquivalente Problem eine Gültige Ausrichtung mit kleinstmöglicher Bezugsmenge zu finden. Genauer gesagt wollen wir folgendes Problem lösen:

Gültige Ausrichtung mit
kleinstmöglicher Bezugsmenge (GAKB)
Eingabe: Ein Graph $G(V, E)$.
Ziel: Eine Menge A, so dass eine Gültige Ausrichtung für G bezüglich A existiert und $|A|$ kleinstmöglich ist.

4.4.1 Terminale in einer Gültigen Ausrichtung

Wir versuchen nun einen ähnlichen Algorithmus, wie in Abschnitt 4.2.1 beschrieben, zu entwickeln um GAKB auf GSP-Graphen zu lösen. Die Grundidee war hierbei einen Dynamic-Programming-Ansatz auf dem Parse-Tree zu verfolgen. Für jeweils beide Kinder eines inneren Knotens des Parse-Trees berechnen wir alle relevanten Lösungen. Durch Terminalidentifikation wollen wir aus diesen die relevanten Lösungen des inneren Knoten generieren. Wir wollen das Konzept der Terminalzustände adaptieren. Welche Terminalzustände müssen wir also für GAKB unterscheiden? Wir wollen einfach die verschiedenen Terminalzustände eines Terminals v in einer GA durch $indeg_{F_u}(v)$ und $outdeg_{F_u}(v)$ unterscheiden:

1. $indeg_{F_u}(v) = 1$ und $outdeg_{F_u}(v) = 0$
2. $indeg_{F_u}(v) = 1$ und $outdeg_{F_u}(v) = 1$
3. $indeg_{F_u}(v) = 0$ und $outdeg_{F_u}(v) \geq 0$

Aufgrund der Definition 4.5 sind keine anderen Möglichkeiten für die Knotengrade der Terminale bezüglich F_u in einer Gültigen Ausrichtung vorhanden. Wir schließen in Fall 3, dass v in der Bezugsmenge enthalten sein muss.
Wenn wir eine optimale Lösung für einen Graphen G_3 kennen, müssen wir uns fragen, wie diese Lösung aussieht, falls wir sie getrennt für die Kinder G_1, G_2 im Parse-Tree von G_3 betrachten. Denn wir wollen die Lösung für G_3 aus Teillösungen für G_1 und G_2 konstruieren. Für jeden Zustand 1-3 haben wir eine gewisse Anzahl von Kanten aus F_u. Falls wir G_3 wieder dekomponieren, fällt jede Kante aus F_u entweder G_1 oder G_2 zu. Wir werden nun jeden Zustand durch ein 2-er Tupel (i, o) repräsentieren. In der ersten Komponente steht dann $indeg_{F_u}(v)$ und in der zweiten $outdeg_{F_u}(v)$. Jetzt wollen wir die einzelnen Kanten aus F_u die pro Zustand vorkommen auf die Kindgraphen G_1 und G_2 verteilen. Für den Zustand $(1, 0)$ wird die Kante aus F_u o.B.d.A G_1 zugesprochen. Dann folgt aber, dass in G_2 keine Kante aus F_u zum Terminal inzident ist. Dies wird dann durch das 2-er Tupel $(0, 0)$ ausgedrückt. Tabelle 4.1 fasst diese Überlegungen für alle

Terminalzustand	Entstandene 2-er Tupel	
(1,0)	(1,0)	(0,0)
(1,1)	(1,1) (1,0)	(0,0) (0,1)
$(0,k)$ $k \geq 0$	$(0,k_1)$	$(0,k_2)$ $k = k_1 + k_2$

Tabelle 4.1: *Terminalzustände und ihre Dekomposition.*

Zustände zusammen. Wir entnehmen der Tabelle, dass durch die Dekomposition neue 2-er Tupel entstanden sind, nämlich $(0,0)$ und $(0,1)$. Diese Tupel müssen wir ebenfalls als neue Terminalzustände für G_1 und G_2 betrachten. Insgesamt müssen wir fünf Zustände aus Tabelle 4.1 für Terminale festhalten.
Zuvor soll bemerkt werden, dass die einfache Modifikation des Algorithmus aus [13] bezüglich der Terminalzustände zur Lösung von GAKB nicht ausreicht. Dies ist klar, da GAKB zu PDS äquivalent ist und wir bereits in Abschnitt 4.2.2 sahen, dass die einfache Adaption für PDS problembehaftet ist. Betrachten wir hierzu Abbildung 4.9.

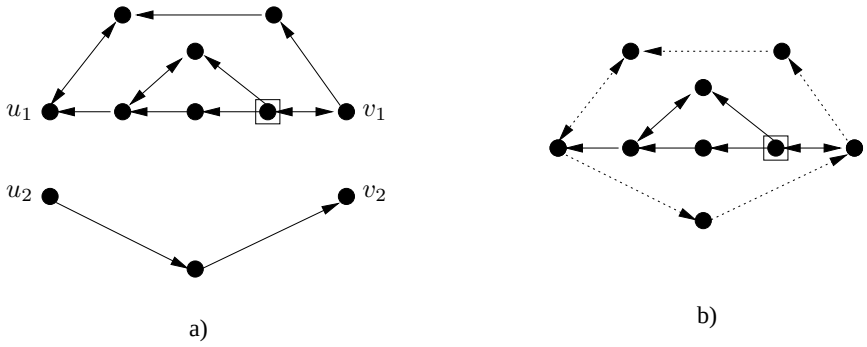


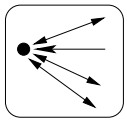
Abbildung 4.9: a*) zwei GSP-Graphen G_1 und G_2. b) Die P-Komposition aus G_1 und G_2. Die gestrichelten Kanten bilden einen alternierenden Kreis.*

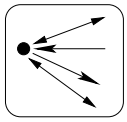
Wir sehen zwei Ausrichtungen für GSP-Graphen $G_1(V_1, E_1, u_1, v_1)$ und $G_2(V_2, E_2, u_2, v_2)$, sodass an den Terminalen jeweils einer der fünf Zustände herrscht. Überträgt man diese beiden Ausrichtungen direkt auf die korrespondierenden Subgraphen in $G_3 = G_1 \odot_P G_2$, so entsteht in der resultierenden Ausrichtung ein alternierender Kreis. Im Unterschied zu PDS können wir bei GAKB genau angeben, in welchem Fall die Komposition zweier Lösungen wieder eine Gültige Ausrichtung ergibt. Dies ist der Fall, wenn die Terminalzustände zueinander passen und wenn kein alternierender Kreis entsteht. Um das Entstehen von alternierenden Kreisen zu verhindern, werden wir später noch ein weiteres Konzept entwickeln müssen. Dieses stellt die hauptsächliche Neuerung im Vergleich zum Algorithmus aus [13] dar. Ebenfalls erhält man dadurch ein besseres Verständnis des 'nicht-lokalen Charakters' von Power Domination.

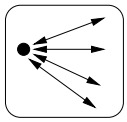
Zustände für Terminale

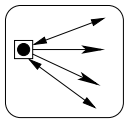
Die Einsichten, die wir im letzten Abschnitt bekamen, wollen wir mit folgenden Definitionen festhalten. Seien G_3 ein GSP-Graph und $D(V, F)$ eine Gültige Ausrichtung bezüglich $A \subseteq V$ für G_3. Seien G_2, G_1 die Kinder von G_3 im Parse-Tree. Betrachten wir $D(V, F)$ eingeschränkt auf G_1 bzw. G_2, so definieren wir für die Terminale von G_1 und G_2 folgende fünf Zustände.

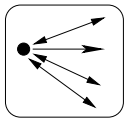
Definition 4.13. *Sei u ein Terminal. Dann definieren wir in Übereinstimmung mit den 2-er Tupel aus Tabelle 4.1 fünf Zustände.*

I)  *(1,0): u ist inzident zu einer eingehenden Kante aus F_u und sonst eventuell zu Kanten aus F_b.*

II)  *(1,1): u ist inzident zu einer ein- und zu einer ausgehenden Kante aus F_u und sonst eventuell zu Kanten aus F_b.*

III)  *(0,0): u ist nur zu Kanten aus F_b inzident.*

IV)  *(0,k): $u \in A$ und u ist eventuell zu mehreren ausgehenden Kanten aus F_u und Kanten aus F_b inzident.*

V)  *(0,1): u ist inzident zu einer ausgehenden Kante aus F_u und sonst eventuell zu Kanten aus F_b.*

Wie wir schon vorhin beobachtet haben, kann es nach Tabelle 4.1 keine weiteren Fälle mehr geben.

Partiell Gültige Ausrichtung

Gilt für ein Terminal u einer der fünf Zustände, dann ist streng genommen die Definition einer Ausrichtung bezüglich einer Knotenmenge A verletzt, denn für jeden Knoten $v \in V \setminus A$ muss $indeg_{F_u}(v) = 1$ gelten. Für die Terminalzustände III und V wird diese Restriktion aber verletzt. Wir müssen nun ein neues Objekt einführen, welches dieses berücksichtigt.

Definition 4.14. *Eine* ℓ-partielle Ausrichtung *(ℓ-PA) $D(V_\eta \cup V_\theta, F)$ ist eine Ausrichtung bezüglich einer Bezugsmenge A, so dass gilt:*

1. $|V_\eta| = \ell$.
2. *Für alle* $v \in V_\eta \setminus A$ *gilt* $indeg_{F_u}(v) \leq 1$ *und* $outdeg_{F_u}(v) \leq 1$.
3. *Für alle* $v \in V$ *gilt:* $indeg_{F_u}(v) = 0 \iff v \in A$ *oder* $v \in V_\eta$

4. *Für alle* $v \in V_\theta \setminus A$ *gilt* $indeg_{F_u}(v) = 1$ *und* $outdeg_{F_u}(v) \leq 1$.

5. *Für alle* $v \in V_\theta \setminus A$ *gibt es einen Observationspfad* $P = (a, w_1 \dots w_q v)$ *mit* $w_1, \dots, w_q \notin A$ *und* $a \in A$ *oder* $a \in V_\eta$.

Definition 4.15. *Eine* ℓ-partiell Gültige Ausrichtung *(ℓ-PGA) ist eine ℓ-partielle Ausrichtung, die keinen alternierenden Kreis enthält.*

Eine ℓ-partiell Gültige Ausrichtung ist also eine Verallgemeinerung einer GA. Aus algorithmischer Sicht sind die Knoten in V_θ diejenigen, die wir im Laufe des Algorithmus nicht mehr, und in V_η diejenigen, die wir in weiteren Schritten noch betrachten. Für die Knoten in V_θ müssen entweder alle Anforderungen einer GA bereits gelten oder es ist uns noch möglich diese mittels eines Knotens aus V_η in einem späteren Schritt sicherzustellen. Da wir die Knoten in V_η noch weiter bearbeiten können, dürfen hier nur die angegebenen Gradbedingungen nicht verletzt werden. In unserem konkreten Fall der GSP-Graphen betrachten wir nur 2-Partiell Gültige Ausrichtungen (2-PGA). Ist dann im Weiteren $D(V_\eta \cup V_\theta, F)$ eine 2-PGA für einen GSP-Graphen $G(V, E, u, v)$, so soll immer $V_\eta = \{u, v\}$ und $V_\theta = V \setminus \{u, v\}$ gelten. Die Bezeichner u und v stellen auch in Hinsicht auf 2-PGA's immer Terminale dar und ein eventuell vorkommender Index bedeutet wieder, dass die Terminale zur 2-PGA mit entsprechendem Index gehören. Analoges gelte auch für 2-partielle Ausrichtungen (2-PA). 2-PGA's sind genau die Objekte, die entstehen wenn wir eine GA für einen GSP-Graphen, der aus der Komposition zweier weiterer GSP-Graphen entsteht, wieder dekomponieren.
Wir wollen uns auf die Zustände der Terminale einer 2-PGA direkt beziehen können. Deshalb legen wir fest:

Definition 4.16. *Ein Knotenzustand für eine 2-PGA* $D(V_\eta \cup V_\theta, F)$ *mit* $V_\eta = \{u, v\}$ *ist eine Abbildung* $z_{D:t} : \{u, v\} \to \{I, II, III, IV, V\}$.

Gilt also für ein Terminal $u \in V_\eta$ dann $z_{D:t}(u) = II$, so heißt dies, dass Zustand II an u herrscht. Die Bezeichnung $z_{D:t}$ soll andeuten, dass wir uns mit t auf die Terminale von D beziehen. Eine 2-PGA besitzt immer zwei Terminalzustände. Wir wollen 2-PGA's anhand dieser beiden Terminalzustände weiter unterscheiden können.

Definition 4.17. *Seien* $a, b \in \{I, \dots, V\}$ *und* $D(V, F)$ *eine 2-PGA. Wenn wir* $D(V, F, a, b)$ *schreiben, so meinen wir eine 2-PGA* $D(V, F)$ *mit Terminalen* u, v, *so dass* $z_{D:t}(u) = a$ *und* $z_{D:t}(v) = b$ *gilt. Analoges definieren wir für eine 2-PA.*

Da es 5 Zustände gibt und jede 2-PGA zwei Terminale besitzt, folgt, dass es 25 potentielle 2-PGA's $D(V, F, a, b)$ für einen GSP-Graphen gibt. Damit wir diese für die inneren Knoten des Parse-Trees auf Bottom-Up-Weise berechnen können, müssen wir für den kleinsten GSP-Graph, eine Kante, alle 2-PGA's $D(V, F, a, b)$ bestimmen. Betrachte hierzu Abbildung 4.10.

Nun kennen wir alle 2-PGA's für die Blätter des Parse-Tree. Im nächsten Schritt müssen wir uns darüber klar werden, wie wir aus den zwei Kindern eines Graphen im Parse-Tree 2-PGA's berechnen können. Dazu halten wir fest, was passiert, wenn eine 2-PGA D_3 für einen Graphen G_3 mit Kindern G_1 und G_2 wieder dekomponiert wird.

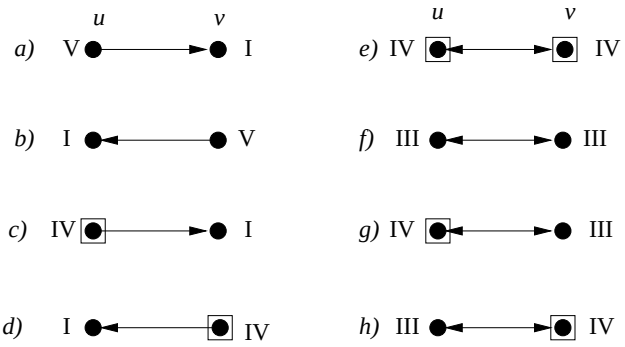


Abbildung 4.10: *In a) − h) sind alle 2-PGA's für eine Kante zu sehen. Da eine Kante selbst auch ein GSP-Graph ist, sind an den Terminalen die Zustände notiert, die sich aufgrund der speziellen Ausrichtung ergeben.*

Beobachtung 4.18. *Seien G_1, G_2 und G_3 GSP-Graphen. Der Graph G_3 entsteht durch Komposition aus G_1 und G_2. Sei $D_3(V_3, E_3, a_3, b_3)$ eine 2-PGA für G_3. Wird D_3 in Entsprechung zu G_3 dekomponiert, so entstehen 2-PGA's $D_1(V_1, E_1, a_1, b_1)$ und $D_2(V_2, E_2, a_2, b_2)$ jeweils für G_1 und G2.*

Beweis. Die Dekomposition betrifft nur die Terminale von G_3. Falls ein Terminal in G_3 durch Identifikation entstand, dann werden die inzidenten Kanten durch die Dekomposition entweder G_1 oder G_2 zugeordnet. Sei u_3 o.B.d.A durch Identifikation aus $d_1 \in V_1$ und $d_2 \in V_2$ entstanden. Dann müssen nach der Dekomposition die zu u_3 inzidenten Kanten entweder d_1 oder d_2 zugeordnet werden. Für $a_3 \in \{I, II, IV\}$ wurde dies in Tabelle 4.1 bereits getan. Mit den gleichen Überlegungen folgt dann aus $a_3 = III$, dass nach der Dekomposition Zustand III an d_1 und Zustand III an d_2 herrschen. Gilt $a_3 = V$, folgt gleichfalls, dass nach der Dekomposition Zustand III an d_1 und Zustand V an d_2 oder umgekehrt herrschen. Da d_1 und d_2 nach der Dekomposition Terminale sind müssen D_1 und D_2 existiereren. □

4.4.2 Komposition

Wie bereits erwähnt wollen wir, indem wir den Parse-Tree eines GSP-Graphen Bottom-Up traversieren, für jeden inneren Knoten die relevanten Lösungen aus den Lösungen der Kinder berechnen. Die relevanten Lösungen sind 2-PGA's. Für die Blätter des Parse-Tree fällt es uns leicht alle möglichen 2-PGA's zu berechnen. Jedoch für einen inneren Knoten müssen wir noch herausfinden, welche 2-PGA's für weitere Rechenschritte nötig sind. 2-PGA's kann man durch ihre Terminalzustände unterscheiden. Es reicht jedoch nicht, wie wir sehen werden, alle 2-PGA's mit kleinstmöglicher Bezugsmenge, die sich in mindestens einem Terminalzustand unterscheiden, zu berechnen. Wir werden sehen, dass wir 2-PGA's noch feiner unterscheiden müssen, damit wir mit Dynamic Programming GAKB lösen können. In diesem Abschnitt werden wir genau die Mechanismen untersuchen, die bei einer Komposition von zwei 2-PGA's auftreten. Wir werden klären wann eine Komposition wieder eine 2-PGA ist und wie wir eine 2-PGA mit bestimmten Terminalzuständen aus den Kompositionskomponenten erstellen können.

Sei nun G_3 ein innerer Knoten des Parse-Tree und G_1 und G_2 seien seine Kinder. Für G_3 wollen wir nun z.B. eine 2-PGA $D(V_3, F_3, I, III)$ bezüglich $A \subseteq V_3$ bestimmen, so dass $|A|$ minimal ist. Nehmen wir weiter an, G_3 entsteht durch P-Komposition aus G_1 und G_2. Haben wir eine 2-PGA $D_1(V_1, F_1, III, III)$ für G_1 und eine weitere $D_2(V_2, F_2, III, III)$ für G_2, so können wir diese weiter verwenden. Indem wir eine P-Komposition der beiden 2-PGA's vornehmen, erhalten wir eine 2-PA mit Terminalzuständen I und III. Wir erhalten keine 2-PGA, da ja durch die Komposition ein alternierender Kreis entstehen kann. Um aus zwei 2-PGA's durch Komposition eine neue 2-PGA zu erhalten, müssen die jeweiligen Terminalzustände 'zueinander passen' und es darf kein alternierender Kreis entstehen.

Terminalzustände und Komposition

Wir wollen nun das Problem der alternierenden Kreise noch ausklammern. Wir fokussieren unser Interesse zunächst darauf, welche zwei 2-PGA's durch die entsprechenden Kompositionen, S1- , S2- oder P-Komposition, in 2-PA's überführt werden. Hier werden wir also entscheiden, welche Terminalzustände in Bezug auf eine Komposition zueinander passen.

Definition 4.19. *Sind D_1, D_2 zwei 2-PA's bzw. 2-PGA's für die GSP-Graphen G_1, G_2, so werden $D_1 \odot_{S1} D_2$, $D_1 \odot_{S2} D_2$ und $D_1 \odot_P D_2$ analog zu $G_1 \odot_{S1} G_2$, $G_1 \odot_S G_2$ und $G_1 \odot_P G_2$ durch die entsprechende Terminalidentifikation definiert.*

S1-Komposition Wenn wir zwei Graphen $G_1(V_1, E_1, u_1, v_1), G_2(V_2, E_2, u_2, v_2)$ miteinander zu G_3 S1-komponieren, identifizieren wir v_1 und u_2. Der neue Knoten d ist dann kein Terminal mehr und folglich gilt in jeder 2-PGA für G_3 dann $d \in V_\theta$. Für ein 2-PGA für G_3 bedeutet dies, dass $indeg_{F_u}(d) = 1$ und $outdeg_{F_u}(d) \leq 1$ gilt oder, dass d in der Bezugsmenge ist. Wenn wir also die entsprechenden 2-PGA's D_1, D_2 S1-komponieren, muss für das Resultat selbiges für d gelten. Dies heißt, dass bestimmte Zustände für v_1 und u_2 nicht kompatibel sind. Gilt $z_{D_1:t}(v_1) = II$ und $z_{D_2:t}(u_2) = I$, so folgt $indeg_{F_u}(d) = 2$. Gilt $z_{D_1:t}(v_1) = III$ und $z_{D_2:t}(u_2) = V$, so folgt $z_{D_3:t}(d) = V$ und somit $indeg_{F_u}(d) = 0$. Beide Male werden die geforderten Eigenschaften einer 2-PGA verletzt. Ist stattdessen $z_{D_1:t}(v_1) = II$ und $z_{D_2:t}(u_2) = III$ der Fall, so gilt $indeg_{F_u}(d) = 1$ und $outdeg_{F_u}(d) = 1$.

Proposition 4.20. *Seien G_1, G_2 GSP-Graphen und $D_1(V_1, F_1, a_1, b_1)$, $D_2(V_2, F_2, a_2, b_2)$ entsprechende 2-PGA's. Dann ist $D_3(V_3, F_3, a_1, b_2) = D_1 \odot_{S1} D_2$ eine 2-PA, falls $b_1 \odot_{S1} a_2 = *$ in Tabelle 4.2 gilt.*

Beweis. Für den neu entstandenen Knoten d gilt entweder, dass er in der Bezugsmenge für D_3 enthalten ist. Dann muss er aber auch in den jeweiligen Bezugsmengen von D_1 und D_2 zu finden sein. Oder es muss in D_3 $indeg_{F_u}(d) = 1$ und $outdeg_{F_u}(d) \leq 1$ gelten. In der Tabelle 4.2 steht dann ein $*$, falls $z_{D_1:t}(v_1)$ und $z_{D_2:t}(u_2)$ diese Bedingungen nicht verletzen. Da $z_{D_3:t}(u_3) = z_{D_1:t}(u_1)$ und $z_{D_3:t}(v_3) = z_{D_2:t}(v_2)$ gilt, folgt die Behauptung. □

$\oplus_{S1}$	I 	II 	III 	IV 	V 
I 	-	-	$*$	-	$*$
II 	-	-	$*$	-	-
III 	$*$	$*$	-	-	-
IV 	-	-	-	$*$	-
V 	$*$	-	-	-	-

Tabelle 4.2: *Ein Strich gibt an, dass eine S1-Komposition der Zeilen- und Spaltenelemente nicht möglich ist. Ein Stern $*$ erlaubt diese jedoch. Gilt z.B. $z(v_1) = I$ und $z(u_2) = III$, dann darf der Knoten $v_1 = u_2$, der durch die S1-Komposition entsteht, zu einem inneren Knoten des entstanden Graphen werden, ohne die Anforderungen an eine 2-partiell Gültige Ausrichtung zu verletzten.*

P-Komposition Seien $G_1(V_1, E_1, u_1, v_1), G_2(V_2, E_2, u_2, v_2)$ GSP-Graphen, die zu G_3 P-komponiert werden. Die Knoten u_1, u_2 und v_1, v_2 werden dann miteinander identifiziert und es entstehen u_3 und v_3. Da u_3 und v_3 wieder Terminale sind, muss in einer 2-PGA D_3 für G_3 dort jeweils einer der fünf Terminalzustände herrschen. Wenn wir dann entsprechende 2-PGA's D_1 und D_2 miteinander P-komponieren, so gibt es wieder inkompatible Terminalzustände, die die geforderten Eigenschaften für eine 2-PGA verletzen. Da u_3 und v_3 wieder Terminale sind, gibt es keine verbotenen Terminalzustände für sie. Gilt beispielsweise $z(u_1) = I$ und $z(u_2) = V$, so folgt $z(u_3) = II$. Jedoch müssen die Gradbedingungen bezüglich der zu u_3 und v_3 inzidenten Kanten aus F_b eingehalten werden.

Proposition 4.21. *Seien G_1, G_2 GSP-Graphen und $D_1(V_1, F_1, a_1, b_1)$, $D_2(V_2, F_2, a_2, b_2)$ entsprechende 2-PGA's. Sei $G_3 = G_1 \odot_P G_2$. Dann ist $D_3(V_3, E_3, a_3, b_3) = D_1 \odot_P D_2$ eine 2-PA, falls $a_1 \odot_P a_2 = a_3$ und $b_1 \odot_P b_2 = b_3$ in Tabelle 4.3 gilt.*

Beweis. Falls u_1 und u_2 in der jeweiligen Bezugsmenge waren, so folgt dies auch für u_3. Ansonsten muss $z(u_3) \in \{I, \ldots, V\}$ gelten. In der Tabelle 4.3 wird festgehalten, für welche Zustände für u_1 und u_2 diese Bedingungen nach der P-Komposition nicht verletzt sind. Analoges gilt für v_1 und v_2. □

$\oplus_P$	I 	II 	III 	IV 	V 
I 	-	-	I 	-	II 
II 	-	-	II 	-	-
III 	I 	II 	III 	-	V 
IV 	-	-	-	IV 	-
V 	II 	-	V 	-	-

Tabelle 4.3: *Ein Strich deutet an, dass eine P-Komposition zwischen den entsprechenden Zeilen- und Spaltenelementen unmöglich ist. Sonst ist der Zustand nach der P-Komposition, der dann am Terminal herrscht, abzulesen. Beispielsweise wenn $z(u_1) = II$ und $z(u_2) = III$ gilt, so heißt dies $z(u_1) = II$ nach der P-Komposition mit $u_1 = u_2$.*

S2-Komposition

Proposition 4.22. *Seien G_1, G_2 GSP-Graphen und $D_1(V_1, F_1, a_1, b_1)$, $D_2(V_2, F_2, a_2, b_2)$ entsprechende 2-PGA's. Sei $G_3 = G_1 \odot_{S_2} G_2$. Dann ist $D_3(V_3, E_3, a_3, b_3) = D_1 \odot_{S2} D_2$ eine 2-PA, falls $b_2 \in \{I, II, IV\}$, $a_1 = a_3$ und $b_1 \odot_P a_2 = b_3$ in Tabelle 4.3 gilt.*

Beweis. Die S2-Komposition unterscheidet sich von der S1-Komposition nur dadurch, dass nicht $v_1 = u_2$ zu einem Nicht-Terminal wird, sondern v_2. Da v_2 nie wieder betrachtet wird, darf dort keine Anforderung an eine 2-PGA verletzt sein. Folglich heißt dies $b_2 \in \{I, II, IV\}$. Der aus $v_1 = u_2$ resultierende Knoten v_3 wird jedoch zum Terminal von D_3. Also muss an v_3 wieder ein gültiger Zustand herrschen, analog zu den Terminalen in der P-Komposition. Deswegen muss $b_1 \odot_P a_2 = b_3$ gelten. Da mit u_1 nichts geschieht, heißt dies $a_1 = a_3$. □

Jetzt ist uns bekannt wie wir ein 2-PA $D(V, F, a, b)$ rekursiv für einen Graphen konstruieren können. Wir werfen abhängig von der Art der Komposition einen Blick in die entsprechende Tabelle. Dort können wir ablesen welche Terminalzustände an den Knoten, die miteinander identifiziert werden, herrschen müssen. Wenn wir diese Terminalzustände kennen, müssen wir nur noch die entsprechenden partiellen Ausrichtungen der Kinder komponieren.

Alternierende Kreise und Komposition

Wie uns zuvor schon aufgefallen ist, können bei der Komposition von zwei 2-PGA's alternierende Kreise entstehen, siehe Abbildung 4.9. Deswegen können wir bis jetzt nur garantieren, dass nach der Komposition eine 2-PA entsteht. Wir werden jetzt untersuchen wie alternierende Kreise durch Komposition entstehen. Wie folgende Proposition zeigt, müssen wir uns nur mit der P-Komposition beschäftigen, um das Entstehen von alternierenden Kreisen zu erklären.

Proposition 4.23. *Seien D_1, D_2 2-PGA's. D_1 und D_2 enthalten insbesondere keine alternierenden Kreise. Dann enthalten auch $D_3 = D_1 \odot_{S1} D_2$ und $D_4 = D_1 \odot_{S2} D_2$ keine alternierenden Kreise.*

Beweis. Nehmen wir an D_3 hat einen alternierenden Kreis $C = (v_1, \ldots, v_n)$. Sei d der Knoten in D_3, der durch die Identifikation zweier Terminale entstanden ist. Da wir wissen, dass D_1 und D_2 keinen alternierenden Kreis besaßen, so muss C neu entstanden sein. Also muss C sowohl durch D_1 als auch durch D_2 laufen. Dann muss aber der Knoten d zweimal in C vorkommen, da d ein Separator ist. Dann ist C aber ein geschlossener Weg und kein Kreis. Widerspruch.
Da die S2-Komposition die gleichen Terminale identifiziert, gelten auch hier die obigen Voraussetzungen für den in D_4 entstehenden Knoten d'. Also ist obige Argumentation auch in diesem Fall gültig. □

Alternierende Pfade Alternierende Kreise entstehen also nur durch die P-Komposition, wie wir durch Abbildung 4.9 und Proposition 4.23 jetzt wissen. Betrachten wir eine 2-PA mit alternierendem Kreis, die durch P-Komposition zweier weiterer 2-PGA's entstanden ist. Machen wir die P-Komposition wieder rückgängig so zerfällt auch der Kreis in zwei Teile. Diese zwei Teile sind Pfade die von einem Terminal zum anderen führen. Durch die P-Komposition dieser beiden Pfade ist also der alternierende Kreis entstanden. Diese Pfade müssen den alternierenden Kreis in ihrer Struktur ähnlich sein. Dies heißt, dass eine Sequenz aus mehreren Kanten aus F_u sich immer mit einer Kante aus F_b abwechselt. Wir definieren dies genauer.

Definition 4.24. *Ein gerichteter Pfad zwischen den Terminalen einer partiellen Ausrichtung bezüglich einer Bezugsmenge A ist alternierend, falls keine zwei Kanten aus F_b inzident sind und kein Knoten in A enthalten ist.*

Wenn wir zwei 2-PGA's, die jeweils einem alternierenden Pfad P_1 bzw. P_2 besitzen, P-komponieren, so muss nicht zwangsläufig ein alternierender Kreis entstehen. Wenn z.B. P_1 und P_2 in der Komposition, also nach der Knotenidentifikation, vom gleichen Terminal starten und somit auch beim gleichen Terminal enden. Dann sprechen wir im Folgenden davon, dass zwei Pfade die gleiche Richtung haben. Oder aber auch wenn beide Pfade zum gleichen Terminal mit einer Kante aus F_b inzident sind, kann daraus kein alternierender Kreis

entstehen. Um entscheiden zu können, welche alternierenden Pfade nach einer P-Komposition einen Kreis bilden können, nehmen wir eine Klassifikation vor. Wir unterscheiden alternierende Pfade in Bezug darauf mit welchen Kanten, entweder solchen aus F_u oder solchen aus F_b, sie zu den Terminalen inzident sind. Denn genau dann wenn die Richtungen von P_1 und P_2 verschieden sind und zu jedem Terminal höchstens eine Kante aus $F_b \cap (E(P_1) \cup E(P_2))$ inzident ist, entsteht ein Kreis. Diese Klassifikation ist fein genug, da wir über den inneren Aufbau der Pfade nichts wissen müssen, außer, dass keine zwei Kanten aus F_b inzident sind. Zu jedem Terminal kann eine Kante aus F_u oder F_b inzident sein. Da wir zwei Terminale betrachten, muss es also insgesamt vier Klassen geben.

Definition 4.25. *Seien v, u Terminale in einem GSP-Graphen. Wir unterteilen die alternierenden Pfade von v nach u bezüglich der Anzahl ihrer Kanten aus F_b, die zu u oder v inzident sind. Sei P nun solch ein Pfad. Er gehört zu einer der folgenden Klassen, falls gilt:*

Klasse 1: $N_E(v) \cap E(P) \in F_u$ *und* $N_E(u) \cap E(P) \in F_u$.

Klasse 2: $N_E(v) \cap E(P) \notin F_u$ *und* $N_E(u) \cap E(P) \in F_u$.

Klasse 3: $N_E(v) \cap E(P) \in F_u$ *und* $N_E(u) \cap E(P) \notin F_u$.

Klasse 4: $N_E(v) \cap E(P) \notin F_u$ *und* $N_E(u) \cap E(P) \notin F_u$.

Alle so auftretenden Klassen sind in Abbildung 4.11 aufgeführt.

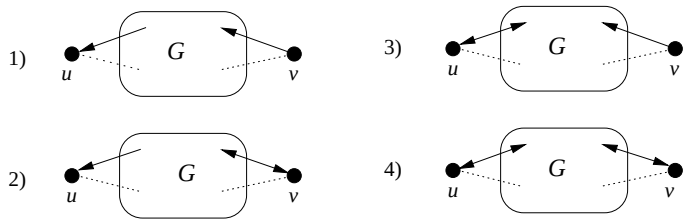


Abbildung 4.11: *Alternierende Pfade werden danach unterschieden, ob die zu den Terminalen inzidenten Kanten aus F_u oder F_b sind.*

Um uns auf alle Pfadklassen zwischen Terminalen v, u für eine 2-PGA D für einen GSP-Graphen $G(V, E, v, u)$ zu beziehen, ziehen wir eine Abbildung $z_{D:p}$ zu Hilfe heran.

Definition 4.26. *Ein Knotenpaarzustand für eine 2-PGA $D(V_\eta \cup V_\theta, F)$ mit $V_\eta = \{u, v\}$ ist eine Abbildung $z_{D:p} : \{(u, v), (v, u)\} \to \mathcal{P}(\{1, \ldots, 4\})$.*

Gilt also z.B. $z_{D:p}(u, v) = \{1, 4\}$, so gibt es mindestens einen Pfad der Klasse 1 und mindestens einen der Klasse 4 von u nach v.
In Abbildung 4.10 sehen wir alle möglichen 2-PGA's für eine Kante. Wir wollen jetzt in diesen die alternierenden Pfade finden. Sobald ein Terminal in der Bezugsmenge A enthalten ist, ist klar, dass kein alternierender Pfad vorliegen kann. Für die restlichen 2-PGA's sind die alternierenden Pfade in Abbildung 4.12 zu sehen.

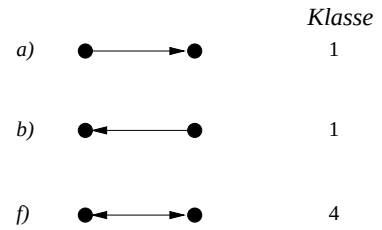


Abbildung 4.12: *Alternierende Pfade der Kante.*

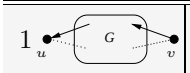
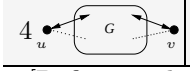
Wie alternierende Kreise entstehen Im vorigen Abschnitt haben wir herausgefunden, dass alternierende Kreise nur durch die P-Komposition entstehen können. Auch nur dann, wenn in den beiden Kompositionskomponenten die geeigneten alternierenden Pfade vorkommen. Wir wollen nun dies im Detail analysieren und die Klassen von alternierenden Pfaden finden, die in einer P-Komposition zusammen alternierende Kreise bilden.

Proposition 4.27. *Seien D_1, D_2 2-PGA's für G_1, G_2 und $D_3 = D_1 \odot_P D_2$. Dann gilt: D_3 enthält einen alternierenden Kreis $\iff$ Es gibt $p_1 \in z_{D_1:p}(u_1, v_1)$ und $p_2 \in z_{D_2:p}(v_2, u_2)$ mit $p_1 \oplus_P p_2 = \dagger$ oder $p_1' \in z_{D_1:p}(v_1, u_1)$ und $p_2' \in z_{D_2:p}(u_2, v_2)$ mit $p_1' \oplus_P p_2' = \dagger$ in Tabelle 4.4.2.*

Beweis. In D_3 ist genau dann ein alternierender Kreis, wenn wir jeweils in D_1 und D_2 alternierende Pfade P_1 und P_2 finden, die verschiedene Richtungen besitzen und wenn keine zwei Kanten aus $F_b \cap (E(P_1) \cup E(P_2))$ in D_3 inzident zu einem Terminal sind. Klar ist, wenn wir einen alternierenden Kreis C in D_3 haben und D_3 wieder dekomponieren, dann zerfällt C in zwei alternierende Pfade P_1 und P_2. Da C ein Kreis ist, führt o.B.d.A. P_1 von u_3 nach v_3 (bzw. u_1 nach v_1) und P_2 von v_3 nach u_3 (bzw, von v_2 nach u_2). P_1 und P_2 haben dann verschiedene Richtungen. Da C ein alternierender Kreis ist, sind keine zwei Kanten aus F_b inzident. Dann sind aber, wenn wir P_1 und P_2 separat in D_3 betrachten, keine zwei Kanten aus $F_b \cap (E(P1) \cup E(P_2))$ inzident. Denn P_1 und P_2 formen in D_3 gerade C.
Seien P_1, P_2 die entsprechenden Pfade in G_1, G_2 der Klassen p_1, p_2. Für P_1 und P_2, gilt das sie an verschiedenen Terminalen beginnen, also verschiedene Richtungen haben. In Tabelle 4.4.2 kann man nun ablesen, ob für zwei alternierende Pfade aus entsprechenden Pfadklassen, mit verschiedenen Richtungen, gilt, dass keine zwei Kanten aus $F_b \cap (E(P_1) \cup E(P_2)$ inzident zu einem Terminal sind. Dann liest man dort $P_1 \oplus_P P_2 = \dagger$ und genau dann entsteht ein alternierender Kreis. Analoges gilt für Pfade P_1' und P_2' der Klassen p_1' und p_2'. □

Wir sind nun vollkommen im Bilde darüber, wann und wie alternierende Kreise entstehen. Nämlich durch die P-Komposition von bestimmten alternierenden Pfaden. Wir wollen nun alternierende Pfade in Hinblick auf die S1-Komposition betrachten.

Wie sich Alternierende Pfade ändern Alternierende Pfade laufen von einem Terminal zum anderen. Wenn wir nun einen Pfad P_1 in einer 2-PGA D_1 und einen weiteren P_2 in D_2 haben, was kann man dann über beide in $D_3 = D_1 \odot_{S_1} D_2$ sagen? Sie müssen auf jeden Fall eine Veränderung erfahren

$\oplus_P$	1 	2	3	4
1 	†	†	†	†
2 	†	†	-	-
3 	†	-	†	-
4 	†	-	-	-

[Definition der Operation $\oplus_P$ für alternierende Pfade]*Gibt das Resultat in Bezug auf alternierende Pfade, die verschiedene Richtungen besitzen, einer P-Komposition aus zwei 2-PGA's $D_1(V,F), D_2(V,F)$ wieder. Ist beispielsweise in D_1 ein alternierender Pfad der Klasse 2 und in D_2 der Klasse 1, so kann man das Resultat durch das zweite Spalten- und erste Zeilenelement ablesen. Ein † steht dafür, dass ein alternierender Kreis und ein '-', dass keiner entsteht.*

haben, da jeweils eines ihrer Terminale in D_3 keines mehr ist. Seien beispielsweise P_1 in der Klasse 3 und P_2 in der Klasse 2. Sei d der Knoten in D_3 der durch Identifikation entstanden ist. P_1 und P_2 sollen die gleiche Richtung haben. Wenn wir nun bei P_2 beginnend bis d gehen und ab dort P_1 beschreiten, so sehen wir, dass dies ein Pfad der Klasse 4 ist. Es ist nur wichtig, dass die Eigenschaften eines alternierenden Pfades an d eingehalten werden. In welcher Klasse sich der entstandene Pfad befindet, kann man an den Anfangs- und Endkanten ablesen.

Proposition 4.28. *Seien D_1, D_2 2-PGA's für G_1, G_2, $G_3 = G_1 \odot_{S1} G_2$ und $D_3 = D_1 \odot_{S_1} D_2$. Dann gilt:*

1. *$k \in z_{D_3:p}(u_3, v_3) \iff$ Es gibt $i \in z_{D_2:p}(u_1, v_1)$ und $j \in z_{D_2:p}(u_2, v_2)$ mit $i \oplus_{S1} j = k$ in Tabelle 4.4.*

2. *$k \in z_{D_3:p}(v_3, u_3) \iff$ Es gibt $i \in z_{D_1:p}(v_2, u_2)$ und $j \in z_{D_2:p}(v_1, u_1)$ mit $i \oplus_{S1} j = k$ in Tabelle 4.4.*

Beweis.

1. Wenn wir einen alternierenden Pfad P_3 der Klasse k von u_3 nach v_3 haben, so muss dieser nach der Dekomposition in alternierende Pfade P_1 von u_1 nach v_1 und P_2 von u_2 nach v_2 zerfallen. Diese müssen natürlich auch alternierende Pfade sein und deshalb auch in eine Klasse einteilbar sein.
   Umgekehrt haben wir in D_1 und D_2 alternierende Pfade P_1 und P_2, die von u_1 nach v_1 bzw. von u_2 nach v_2 führen. Wenn am Knoten, der durch Identifikation entstanden ist, keine zwei Kanten aus $F_b \cap (E(P_1) \cup E(P_2))$ inzident sind, so bilden P_1 und P_2 durch Konkatenation einen neuen alternierenden Pfad P_3 in D_3, der von u_3 nach v_3 führt. P_3 ist dann natürlich wieder klassifizierbar.
   Tabelle 4.4 fasst diese Überlegungen für alle Pfadklassen, die P_1, P_2 und P_3 annehmen können, zusammen.

2. Analog wie 1. □

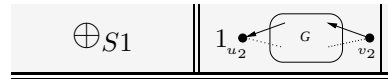
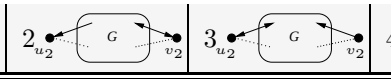
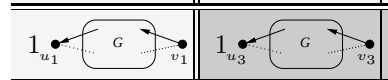
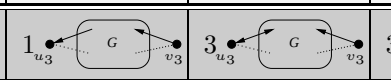
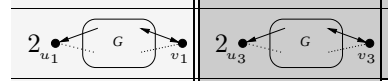
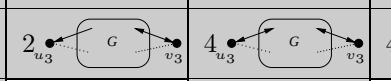
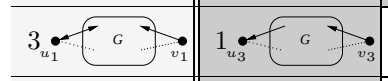
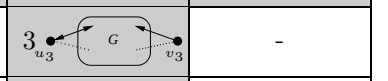
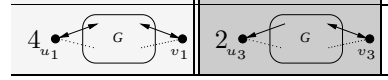
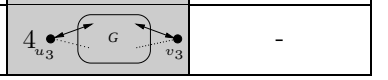
$\oplus_{S1}$				
				
				
		-		-
		-		-

Tabelle 4.4: *Gibt das Resultat einer S1-Komposition aus 2-PGA's $D_1(V,F), D_2(V,F)$ in Bezug auf alternierende Pfade wieder. Sind beispielsweise in D_1 ein alternierender Pfad P_1 der Klasse 2 und in D_2 ein weiterer P_2 der Klasse 3 enthalten, die beide dieselbe Richtung haben, so kann man das Resultat der Konkatenation von P_2 an P_1 durch das zweite Spalten- und dritte Zeilenelement ablesen. Ein '-' bedeutet, dass durch die S1-Komposition der beiden alternierenden Pfade P_1 und P_2 kein alternierender Pfad durch die entsprechende Konkatenation entsteht. Eine Nummer hingegen gibt die Klasse des entstandenen alternierenden Pfades an. Im obigen Fall lesen wir für P_1 und P_2 eine 4.*

Durch die S1-Komposition entstehen zwar keine alternierenden Kreise, aber es können ganze Pfade verschwinden oder neu entstehen. Jetzt müssen wir die S2- und P-Komposition hinsichtlich dessen genauer untersuchen.

Proposition 4.29. *Seien D_1, D_2 2-PGA's für G_1, G_2, und es gelte $G_3 = G_1 \odot_P G_2$, $D_3 = D_1 \odot_P D_2$ und $G_4 = G_1 \odot_{S2} G_2$, $D_4 = D_1 \odot_{S2} D_2$.*

1. $z_{D_3:p}(u_3, v_3) = z_{D_1:p}(u_1, v_1) \cup z_{D_2:p}(u_2, v_2)$ *und*
   $z_{D_3:p}(v_3, u_3) = z_{D_1:p}(v_1, u_1) \cup z_{D_2:p}(v_2, u_2)$
2. $z_{D_4:p}(u_4, v_4) = z_{D_1:p}(u_1, v_1)$ *und* $z_{D_4:p}(v_4, u_4) = z_{D_4:p}(v_1, u_1)$

Beweis.

- ✯ Die 2-PGA D_3 hat die Terminale u_3 und v_3. Alle alternierenden Pfade die in D_1 zwischen u_1 und v_1 verlaufen, verlaufen in D_3 zwischen u_3 und v_3. Dasselbe gilt für alle alternierenden Pfade in D_2. Also sind in $z_{D_3:P}(u_3, v_3)$ alle Pfadklassen vorhanden, die auch in $z_{D_1:p}(u_1, v_1)$ und $z_{D_2:p}(u_2, v_2)$ vorkommen. Analoges gilt für $z_{D:p}(v_3, u_3)$
- ✯ Die 2-PGA D_4 hat die gleichen Terminale wie D_1. D_4 enthält also alle alternierenden Pfade zwischen u_1 und v_1. Dies sind aber genau alle alternierenden Pfade, die auch D_1 besitzt. Also gilt $z_{D_4:p}(u_4, v_4) = z_{D_1:p}(u_1, v_1)$. Analoges gilt für $z_{D_4:p}(v_4, u_4)$. □

Die S2- und P-Komposition verändern die vorliegenden alternierenden Pfade also nicht. Wir haben jetzt alle notwendigen Informationen in Bezug auf alternierende Kreise und Pfade. Wir wissen durch welche alternierenden Pfade und

durch welche Komposition dieser Pfade ein alternierender Kreis erzeugt wird. Uns ist auch der Mechanismus klar, wie mittels S1-Komposition alternierende Pfade gebildet werden. Mit Hilfe dieses Wissens wollen wir nun den angekündigten Algorithmus formulieren.

4.4.3 Dynamisches Programmieren

Fassen wir unsere bisherigen Überlegungen zusammen. Grundsätzlich wollen wir für einen Graphen, der einen inneren Knoten im Parse-Tree repräsentiert, alle relevanten Lösungen aus den Lösungen seiner Kindern berechnen. Wichtig ist nun, dass wir uns klar machen, welche die relevanten Lösungen sind. Denn wir verfolgen einen Dynamic-Programming-Ansatz und wollen pro innerem Knoten des Parse-Trees nur eine konstante Anzahl von Lösungen speichern müssen. In einem ersten Schritt hatten wir uns überlegt, dass wir pro innerem Knoten alle 2-PGA's $D(V, F, a, b)$ mit $a, b \in \{I, II, III, IV, V\}$ und kleinstmöglicher Bezugsmenge speichern wollen. Dann entdeckten wir aber, dass durch eine P-Komposition von 2-PGA's alternierende Kreise entstehen können. Und dies war zurückzuführen auf alternierende Pfade in den beiden Kompositionskomponenten.
Dies heißt nun, dass wir 2-PGA's nicht nur bezüglich ihrer Terminalzustände unterscheiden müssen, sondern auch aufgrund ihrer Pfadklassen die sie enthalten.
Wir wollen hier ein Beispiel anführen. Für einen GSP-Graphen $G(V, E, u, v)$ haben wir zwei 2-PGA's $D_1(V, F, a, b), D_2(V, F, a, b)$ berechnet. D_1 ist eine 2-PGA bezüglich A_1 und D_2 bezüglich A_2. Es gelte nun $|A_1| < |A_2|$. Dann müssen wir aus Sicht der Terminalzustände nur D_1 speichern und können D_2 verwerfen. Legen wir nun jedoch den Fall zugrunde, dass D_1 zwei alternierende Pfade von u_1 nach v_1 besitzt, nämlich jeweils einen aus der Klasse 2 und 3, und D_2 nur einen alternierenden Pfad der Klasse 3 von u_1 nach v_1 besitzt. Verwerfen wir nun D_2 und speichern nur D_1, so kann es sein, dass gerade aufgrund des zusätzlichen alternierenden Pfades der Klasse 2 in einer späteren P-Komposition ein alternierender Kreis entsteht. Sei zum Beispiel D' eine 2-PGA für einen weiteren Graphen $G'(V', E', u', v')$ und es gilt $G'' = G \odot_P G'$. Wenn nun D' einen Pfad der Klasse 2 von v' nach u' enthält, so enthält $D_1 \odot_P D'$ einen alternierenden Kreis und kann somit keine Lösung für G'' mehr sein. $D_2 \odot_P D'$ hingegen enthält keinen alternierenden Kreis, kann also durchaus noch zu einer Lösung beitragen.
Das obige Beispiel zeigt uns, dass wir 2-PGA's nicht nur durch ihre Terminalzustände unterscheiden müssen, sondern auch durch die alternierenden Pfade, die sie enthalten. Genauer gesagt, müssen wir sie bezüglich ihrer enthaltenen Pfadklassen und deren Richtung unterscheiden. Denn, ob ein oder mehrere Pfade der gleichen Klasse und Richtung in einer partiell Gültigen Ausrichtung vorliegen, ist für die Entstehung eines alternierenden Kreises unerheblich. Obiges Szenario kann für alle Teilmengen Q_1, Q_2, die alternierende Pfadklassen darstellen, auftreten, nicht nur für $Q_1 = \{2, 3\}$ und $Q_2 = \{3\}$. Jetzt soll D_1 alle Pfadklassen aus Q_1 und D_2 alle aus Q_2 besitzen. Verwerfen wir nun D_2 und speichern nur D_1, so kann für alle Pfadklassen $K \in Q_1 \setminus Q_2$ ein Kreis in einer späteren P-Komposition entstehen, der in der P-Komposition mit D_2 nicht entstanden wäre.
Selbst wenn $Q_1 \subseteq Q_2$ der Fall ist, kann D_2 relevant sein, wenn $|A_2| < |A_1|$ gilt. Denn es muss ja nicht zwangsläufig aufgrund einer Pfadklasse aus $Q_2 \setminus Q_1$ ein

alternierender Kreis entstehen. Und dann wäre D_2 günstiger als D_1.
Die Folge hieraus ist, dass wir für jeden Knoten im Parse-Tree, der zu einem Teilgraphen $G(V, E, u, v)$ korrespondiert, alle kleinstmöglichen 2-PGA's $D(V, F, a, b)$ mit $a, b \in \{I, \ldots, V\}$ speichern, sodass sie für alle $s_1 \in S_{uv}$ und $s_2 \in S_{vu}$, $S_{uv}, S_{vu} \subseteq \{1, \ldots, 4\}$, mindestens einen Pfad der Klasse s_1 bzw. s_2 von u nach v bzw. von v nach u enthalten.

Aufbau und Initialisierung der Tabelle

Wir wissen nun welche Lösungen wir für einen Graphen $G(V, E, u, v)$, der ein innerer Knoten im Parse-Tree ist, speichern müssen. Um diese genauer beschreiben zu können, brauchen wir folgende Definition.

Definition 4.30. *Ein Zustand für eine 2-PGA $D(V_\eta \cup V_\theta, F)$ mit $V_\eta = \{u, v\}$ ist eine Abbildung $z_D : \{u, v, (u, v), (v, u)\} \to \{I, \ldots, V\} \cup \mathcal{P}(\{1, \ldots, 4\})$, wobei gilt:*

$$z_D(x) = \begin{cases} z_{D:t}(x) : x \in \{u, v\} \\ z_{D:p}(x) : x \in \{(u, v), (v, u)\} \end{cases}$$

Ein Zustand z_D hat einen Definitionsbereich von vier Elementen. Wir werden z_D, wo es nötig ist, durch das Bild dieser Elemente darstellen. Dies geschieht durch ein 4-er Tupel (a, b, S_{uv}, S_{vu}), sodass $z_D(u) = a$, $z_D(v) = b$, $z_D(u, v) = S_{uv}$ und $z_D(v, u) = S_{vu}$ gilt.

Wenn wir im Folgenden davon sprechen, dass eine 2-PGA einen Zustand respektiert, dann bedeutet dies, dass sie die entsprechenden Knotenzustände und Pfadklassen aufweist.

Wir stellen eine Tabelle T_G für jeden Graphen $G(V, E, u, v)$ zu einem inneren Knoten auf, die mit einem Zustand z indexiert wird. Dabei soll für einen Eintrag Folgendes gelten:

$$T_G(z) :=$$
$$\min_{A \subseteq V} \left\{ |A| : \begin{array}{l} \exists \text{ 2-PGA } D(V, F, a, b) \text{ bzgl. } A, z(u) = a, z(v) = b, \ \forall s_1 \in z(u, v), \\ s_2 \in z(v, u) \ \exists \text{ alt. Pfad vom Typ } s_1 \text{ bzw. } s_2 \text{ zw. } u, v \text{ bzw. zw. } v, u \end{array} \right\}$$

Damit speichern wir jede relevante Information bezüglich der möglichen Lösungen eines Graphen G. Existiert für einen Eintrag die geforderte 2-PGA nicht, so erhält der Eintrag den Wert ∞. Da es zwei Terminale, fünf Terminalzustände und 16 verschiedene Kombinationen für die Pfadklassen zwischen den Terminalen gibt, speichern wir für jeden inneren Knoten des Parse-Tree $5 \cdot 5 \cdot 16 \cdot 16 = 6400$ Einträge. Wir müssen noch die Initialisierung von T vornehmen. Das bedeutet, dass wir für jedes Blatt des Parse-Tree die entsprechenden Einträge für eine Kante zu vollziehen haben. Für eine Kante kennen wir bereits alle 2-PGA's aus Abbildung 4.10. Aus Abbildung 4.12 wissen wir welche Klassen von alternierenden Pfaden sie enthalten. Für den einfachsten GSP-Graphen, eine Kante $\{u, v\}$, können wir dann eine Initialisierung vornehmen.

a) $T((V, I, \{1\}, \emptyset)) = 0$ b) $T((I, V, \emptyset, \{1\})) = 0$
c) $T((IV, I, \emptyset, \emptyset)) = 1$ d) $T((I, IV, \emptyset, \emptyset)) = 1$
e) $T((IV, IV, \emptyset, \emptyset)) = 2$ f) $T((III, III, \{3\}, \emptyset)) = 0$
f) $T((III, III, \emptyset, \{3\})) = 0$ g) $T((IV, III, \emptyset, \emptyset)) = 1$
h) $T((III, IV, \emptyset, \emptyset)) = 1$

Die jeweils vorangestellten Buchstaben beziehen sich direkt auf die partiell Gültigen Ausrichtungen für eine Kante in Abbildung 4.10. Alle Einträge, die nicht aufgeführt wurden, werden auf ∞ gesetzt. Der Buchstabe f) tritt zweimal auf, da ein alternierender Pfad, der nur aus einer Kante aus F_b besteht, die beiden Terminale in beide Richtungen verbindet.

Tabelle für innere Knoten des Parse Tree

Wir haben nun für die Blätter des Parse-Trees die entsprechenden Einträge in die Tabelle T vorgenommen. Wir haben uns somit alle relevanten Lösungen für die kleinsten Bestandteile eines Graphen notiert. Wie können wir nun für einen inneren Knoten alle relevanten Lösungen aus den Lösungen seiner Kinder im Parse-Tree generieren? Dies ist natürlich auch von der Komposition abhängig, die an diesem Knoten vorgenommen wird. G_3 geht aus der Komposition von G_1 und G_2 hervor. Um den Eintrag $T_{G_3}(z_{D_3})$ für einen 2-PGA-Zustand z_{D_3} zu bestimmen, müssen wir z_{D_1}, z_{D_2} finden, sodass gilt:

- ✯ Die Komposition zweier 2-PGA's D_1 und D_2, die z_{D_1} und z_{D_2} jeweils respektieren, enthält keinen alternierenden Kreis.
- ✯ In der Komposition von D_1 und D_2 verletzt kein Terminalknoten die Anforderungen an eine 2-PGA.
- ✯ Die Komposition von D_1 und D_2 muss z_{D_3} respektieren.

Abhängig von der Komposition geben wir nun an, wie wir die Einträge für Graphen, die zu inneren Knoten des Parse-Trees gehören, berechnen.
Um eine 2-PGA für G_3 berechnen zu können, müssen wir insbesondere alle Pfadklassen in D_1 und D_2 P- bzw. S1-komponieren. Wir müssen beispielsweise alle Pfadklassen aus $z_{D_1}(u_1, v_1)$ mit denen aus $z_{D_2}(u_2, v_2)$ S1-komponieren. Damit wir in beiden Fällen der Komposition Notation verkürzen können, definieren wir :

Definition 4.31. *Seien* $J, K \subseteq \{1, \ldots, 4\}$

1. $J \triangle_P K = \begin{cases} \dagger : \textit{falls es } j \in J, k \in K \textit{ gibt mit } j \oplus_P k = \dagger \textit{ in Tabelle 4.4.2} \\ 1 : \textit{sonst} \end{cases}$
2. $J \triangle_{S1} K = \{j \oplus_{S1} k \mid j \in J, k \in K \textit{ und } j \oplus_{S_1} k \neq - \textit{ in Tabelle 4.4}\}$

Sei G_3 durch die Komposition von zwei Graphen G_1 und G_2 entstanden. Für einen beliebigen Zustand z_{D_3} für G_3 müssen wir wissen, welche Zustände z_{D_1} und z_{D_2} geeignet sind. Es muss also bekannt sein, welche 2-PGA's D_1, D_2, die den Restriktionen von z_{D_1} und z_{D_2} genügen und zu G_1, G_2 gehören, wir komponieren können, um eine weitere 2-PGA D_3 mit Zustand z_{D_3} zu erhalten. Dazu müssen bestimmte Anforderung an die Terminalzustände und Pfadklassen in D_1 und D_2 erfüllt sein. Beginnen wir mit den Terminalzuständen.

Definition 4.32. *Seien z_3, z_1 und z_2 Zustände. Dann definieren wir für jede Komposition:*

1. $Z_{S1}^{z_3} = \{(z_1, z_2) \mid z_1(v_1) \oplus_{S1} z_2(u_2) = *, z_3(u_3) = z_1(u_1), z_3(v_3) = z_2(v_2)\}$
2. $Z_P^{z_3} = \{(z_1, z_2) \mid z_1(u_1) \oplus_P z_2(u_2) = z_3(u_3), z_1(v_1) \oplus_P z_2(v_2) = z_3(v_3)\}$
3. $Z_{S2}^{z_3} = \{(z_1, z_2) \mid z_1(v_1) \oplus_P z_2(u_2) = z_3(v_3), z_2(v_2) \in \{I, II, IV\}, z_1(u_1) = z_3(u_3)\}$

Gilt $(z_1, z_2) \in Z_P^{z_3}$, so heißt dies, wenn wir 2-PGA's D_1, D_2, die jeweils z_1, z_2 respektieren, P-komponieren, dass wir D_3 erhalten, welches z_3 bezüglich der Terminalzustände respektiert. Dies folgt aus Proposition 4.21. Analoges folgt für $Z_{S1}^{z_3}$ bezüglich der S1-Komposition mit Proposition 4.20 und für $Z_{S2}^{z_3}$ bezüglich der S2-Komposition mit Proposition 4.22.
Allerdings ist noch nicht garantiert, dass die Komposition von D_1 und D_2 den Zustand z_3 in Bezug auf die Pfadklassen respektiert. Es ist also nicht sicher, ob genau alle Pfadklassen vorkommen, die von z_3 verlangt werden, und ob kein alternierender Kreis vorkommt. Wir müssen jetzt klären, welche Zustände z_1, z_2 nach der Komposition auch die Pfadeigenschaften von z_3 respektieren.

Definition 4.33. *Seien z_3, z_1 und z_2 Zustände. Dann definieren wir für jede Komposition:*

1. $A_{S1}^{z_3} = \{(z_1, z_2) \mid z_3(u_3, v_3) = z(u_1, v_1) \triangle_{S1} z(u_2, v_2), z_3(v_3, u_3) = z_2(v_2, u_2) \triangle_{S1} z_1(v_1, u_1)\}$
2. $A_P^{z_3} = \{(z_1, z_2) \mid z_1(u_1, v_1) \triangle_P z_2(v_2, u_2) \neq \dagger, z_1(v_1, u_1) \triangle_P z_2(u_2, v_2) \neq \dagger$
   $z_3(u_3, v_3) = z_1(u_1, v_1) \cup z_2(u_2, v_2), z_3(v_3, u_3) = z_1(v_1, u_1) \cup z_2(v_2, u_2)\}$
3. $A_{S2}^{z_3} = \{(z_1, z_2) \mid z_3(u_3, v_3) = z_1(u_1, v_1), z_3(v_3, u_3) = z_1(v_1, u_1)\}$

Mit Proposition 4.27 sehen wir, dass für alle $(z_1, z_2) \in A_P^{z_3}$ kein alternierender Kreis in der P-Komposition D_3 aus 2-PGA's D_1, D_2, die z_1, z_2 respektieren, vorkommen kann. Mit Proposition 4.29 respektiert D_3 den Zustand z_3 bezüglich der Pfadklassen. Mit Proposition 4.28 folgern wir Gleichartiges für die Pfadklassen bezüglich der S1- bzw. S2-Komposition, wenn $(z_1, z_2) \in A_{S1}^{z_3}$ bzw. $(z_1, z_2) \in A_{S2}^{z_3}$ gilt. Mit Proposition 4.23 folgt für diese Fälle, dass kein alternierender Kreis in der Komposition entsteht.
Um für einen GSP-Graph, der zu einem inneren Knoten in einem Parse-Tree korrespondiert, die Einträge zu bestimmen, hilft uns nun das folgenden Theorem.

Theorem 4.34. *Sei G_3 ein zu einem inneren Knoten des Parse-Trees gehöriger Graph. Wir können für alle Zustände z_3 die Tabelleneinträge $T_{G_3}(z_3)$ aus den Tabelleneinträgen der Kinder G_1 und G_2 mit nachfolgendem Schema berechnen:*

Falls $G_3 = G_1 \odot_P G_2$,
$$T_{G_3}(z_3) = \min_{(z_1,z_2) \in Z_P^{z_3} \cap A_P^{z_3}} (T_{G_1}(z_1) + T_{G_2}(z_2) - b)$$

Falls $G_3 = G_1 \odot_{S1} G_2$,
$$T_{G_3}(z_3) = \min_{(z_1,z_2) \in Z_{S1}^{z_3} \cap A_{S1}^{z_3}} (T_{G_1}(z_1) + T_{G_2}(z_2) - b)$$

Falls $G_3 = G_1 \odot_{S2} G_2$,
$$T_{G_3}(z_3) = \min_{(z_1,z_2) \in Z_{S2}^{z_3} \cap A_{S2}^{z_3}} (T_{G_1}(z_1) + T_{G_2}(z_2) - b)$$

wobei b die Anzahl der durch Identifikation entstandenen Knoten t zählt für die $z_1(t) = IV$ *und* $z_2(t) = IV$ *gilt.*

Beweis. Klar ist aus dem vorigen Abschnitt, dass aus der Komposition von 2-PGA's D_1 und D_2, die jeweils (z_1, z_2) respektieren mit $(z_1, z_2) \in Z_P^{z_3} \cap A_P^{z_3}$, $(z_1, z_2) \in Z_{S1}^{z_3} \cap A_{S1}^{z_3}$ oder $(z_1, z_2) \in Z_{S2}^{z_3} \cap A_{S2}^{z_3}$, je nach Komposition eine 2-PGA D_3 entsteht, die z_3 respektiert. Der Eintrag $T_{G_3}(z_3)$ ergibt sich wenn wir die beiden Größen der Bezugsmengen von D_1 und D_2 addieren und eventuell die Anzahl b der Knoten abziehen, die Teil beider Bezugsmengen sind.
Zu zeigen ist noch, dass dieser Eintrag auch minimal ist. Haben wir aber eine kleinstmögliche 2-PGA D_3' die z_3 respektiert, so zerfällt diese durch Dekomposition in zwei 2-PGA's D_1' und D_2'. D_1' und D_2' respektieren dann Zustände z_1' und z_2' für die $(z_1', z_2') \in Z_P^{z_3} \cap A_P^{z_3}$, $(z_1', z_2') \in Z_{S1}^{z_3} \cap A_{S1}^{z_3}$ oder $(z_1', z_2') \in Z_{S2}^{z_3} \cap A_{S2}^{z_3}$ je nach Komposition gilt. Somit haben wir auch diese beiden Zustände bei der Berechnung des Eintrags von $T_{G_3}(z_3)$ betrachtet und deswegen ist dieser auch minimal. □

Algorithmus 3 ist dann der aus Theorem 4.34 abgeleitete Algorithmus zur Lösung von PDS auf GSP-Graphen.

Korrektheit und Laufzeit

Theorem 4.35. *Der Algorithmus 3 berechnet eine kleinstmögliche Power Dominating Set für einen Graph G.*

Beweis. Algorithmus 3 berechnet eine Gültige Ausrichtung mit kleinstmöglicher Bezugsmenge. Dies ist nach Theorem 4.12 äquivalent dazu eine kleinstmögliche Power Dominating Set zu berechnen. Nach der Berechnung des Parse-Trees $PT(G)$ von G, werden die Einträge in T für dessen Blätter korrekt wie in Abbildung 4.10 initialisiert. Mit Theorem 4.34 wissen wir, dass die Einträge für innere Knoten in T richtig berechnet werden. Im letzten Schritt wird dann die optimale Lösung aus den Einträgen in T für die Wurzel von $PT(G)$ berechnet. Da wir eine Gültige Ausrichtung mit kleinstmöglicher Bezugsmenge suchen, sind alle 2-PGA's, die mindestens ein Terminal haben an dem Zustand III oder V herrscht, zu verwerfen. Aus den übrigen Einträgen wird korrekt der kleinstmögliche ausgewählt. □

Theorem 4.36. *Algorithmus 3 hat eine Laufzeit von* $O(|V|)$.

Beweis. Die Konstruktion eines Parse-Trees $PT(G)$ in Punkt 1 kann mittels des in [13, 20] vorgeschlagenen Algorithmus in $O(|V|)$ Zeit vollzogen werden. Da in $PT(G)$ $O(|E|)$ Blätter vorhanden sind, kostet die Initialisierung in Punkt 2 $O(|E|)$ Schritte. Die Anzahl der Iterationen in Punkt 3 ist $O(|E|)$, da $PT(G)$ ein Binärbaum ist und daher $O(|E|)$ innere Knoten besitzt. Die Berechnungen, die in jeder Iteration in Punkt 3 vollzogen werden, benötigen nur konstante Zeit. Dies sieht man daran, dass es pro innerem Knoten $5 \cdot 5 \cdot 16 \cdot 16 = 6400$ Einträge gibt und wir alle Kombinationen dieser Einträge für die Kinder eines inneren Knotens betrachten müssen. Da es höchstens 40960000 Kombinationen gibt, ist die Zeitkomplexität von Punkt 3 $O(|E|)$. Folglich ist die Gesamtzeitkomplexität von Algorithmus 3 $O(|E|)$. Da jeder GSP-Graph planar ist, ist die letztendliche Zeitkomplexität $O(|V|)$. □

Algorithm 3 *Berechnet die Größe der Bezugsmenge einer Gültigen Ausrichtung für einen GSP-Graph*

Eingabe: Ein zusammenhängender GSP-Graph $G(V, E)$.

Ausgabe: Die kleinstmögliche Größe von $A \subseteq V$, so dass eine Gültige Ausrichtung bezüglich A für G existiert.

1. Berechne $PT(G)$. In U befinden sich die Blätter von $PT(G)$.
2. **for all** $G(V, E, u, v) \in U, \alpha, \beta \in \{I, \ldots, V\}, S_{uv}, S_{vu} \subseteq \{1, \ldots, 4\}$ **do**
   Initialisiere $T_G(\alpha, \beta, S_{uv}, S_{vu})$.
   end for
3. Traversiere $PT(G)$ auf Bottom-Up-Weise. Für jeden inneren Knoten der für einen Teilgraphen G_3 steht, berechnen wir für alle $z_3 = (\alpha, \beta, S_{uv}, S_{vu})$, $\alpha, \beta \in \{I, \ldots, V\}, S_{uv}, S_{vu} \subseteq \{1, \ldots, 4\}$, den Eintrag $T_{G_3}(z_3)$ aus den Tabelleneinträgen des linken und rechten Kindes G_1 und G_2, je nach Komposition:
   for all $z_3 = (\alpha, \beta, S_{uv}, S_{vu})$ **do**
   if $G_3 = G_1 \odot_P G_2$ **then**
   $T_{G_3}(z_3) = \min_{(z_1,z_2) \in Z_P^{z_3} \cap A_P^{z_3}} (T_{G_1}(z_1) + T_{G_2}(z_2) - b)$
   end if
   if $G_3 = G_1 \odot_{S1} G_2$ **then**
   $T_{G_3}(z_3) = \min_{(z_1,z_2) \in Z_{S1}^{z_3} \cap A_{S1}^{z_3}} (T_{G_1}(z_1) + T_{G_2}(z_2) - b)$
   end if
   if $G_3 = G_1 \odot_{S2} G_2$ **then**
   $T_{G_3}(z_3) = \min_{(z_1,z_2) \in Z_{S2}^{z_3} \cap A_{S2}^{z_3}} (T_{G_1}(z_1) + T_{G_2}(z_2) - b)$
   end if
   end for
4. Rückgabe:
   $\min(\{T_G(\alpha, \beta, S_{uv}, S_{vu}) \mid \alpha, \beta \in \{I, II, IV\}, S_{uv}, S_{vu} \subseteq \{1, \ldots, 4\}\})$

Kapitel 5

Power Domination auf Graphen mit beschränkter Baumweite

Wir wollen uns nun noch einen Schritt weiter weg von Bäumen entfernen. Wir betrachten jetzt Graphen mit beschränkter Baumweite. Die prinzipielle Frage, die sich stellt, ist inwiefern ein Graph einem Baum ähnelt. Die Motivation hinter dieser Frage ist jene, dass viele Probleme, die auf allgemeinen Graphen schwierig zu lösen sind, für Bäume vergleichsweise leicht zu lösen sind. Beispielsweise können wir Vertex Cover, Dominating Set und, wie wir bereits gezeigt haben, Power Dominating Set in linearer Zeit lösen, wenn wir diese Probleme auf Bäume einschränken. Es ist dann natürlich naheliegend zu fragen, wie unterschiedlich ein Graph zu einem Baum sein darf, ohne die Eigenschaften zu verlieren, die es so einfach machen das gegebene Problem zu lösen. Aus diesem Grund wurde das Konzept der Baumzerlegung und der damit verbundene Begriff der Baumweite von Robertson und Seymour [19] eingeführt. Die Baumweite ist eine Art Maß inwiefern ein Graph einem Baum ähnelt. Es gelingt hiermit eine Art Kompromiss zwischen der allgemeinen Definition von Graphen und der algorithmischen Einfachheit von Bäumen zu finden. Im folgenden werden wir die relevanten Konzepte einführen und angeben, wie Baumzerlegungen für die Entwicklung eines Fixed-Parameter-Algorithmus verwendet werden können, um PDS zu lösen.

5.1 Baumzerlegungen

Ein wichtiges Konzept, welches wir für unseren Algorithmus verwenden wollen und nun vorstellen, ist die Baumzerlegung wie sie beispielsweise in [5, 14] beschrieben ist. Zusätzlich werden wir einige grundsätzliche Eigenschaften von Baumzerlegungen beweisen und eine besonders einfache Form von Baumzerlegungen betrachten. Beides werden wir für die Entwicklung des Algorithmus brauchen.

Definition 5.1. *Sei $G(V, E)$ ein Graph. Eine Baumzerlegung von G ist ein Paar $\langle\{X_i \mid i \in I\}, T\rangle$, wobei jedes X_i eine Teilmenge von V ist, und T ein Baum ist, dessen Knoten die Elemente aus I sind. Die folgenden Eigenschaften müssen dann gelten:*

1. $\bigcup_{i \in I} X_i = V$.
2. *Für jede Kante $\{u, v\} \in E$ gibt es ein $i \in I$, so dass $\{u, v\} \subseteq X_i$ gilt.*
3. *Für alle $i, j, k \in I$ gilt, falls j auf dem Pfad zwischen i und k in T liegt, $X_i \cap X_k \subseteq X_j$.*

Die X_i nennen wir Beutel.
Die Weite *von $\langle\{X_i \mid i \in I\}, T\rangle$ ist* $\max\{|X_i| \mid i \in I\} - 1$. *Die* Baumweite *von G ist das kleinste k, sodass G eine Baumzerlegung der Weite k besitzt.*

Da unser Algorithmus extensiv von der Baumzerlegungsstruktur Gebrauch macht, ist es von großer Wichtigkeit, dass wir für einen Graphen eine Baumzerlegung effizient berechnen können. Herauszufinden ob ein gegebener Graph Baumweite k oder nicht hat, ist damit einer der hauptsächlichen Eckpfeiler, um fixed-parameter tractability nachzuweisen. Bodlaender [4] zeigt, dass dieses im allgemeinen **NP**-vollständige Problem (S. Arnborg, D. G. Corneil [2]), fixed-parameter tractable ist. Für konstante Baumweite k gibt er einen Linearzeit-Algorithmus an, dessen konstanter Faktor exponentiell von k abhängt.
Eine Baumzerlegung mit einer ganz besonders einfachen Struktur wird mit der folgenden Definition vorgestellt.

Definition 5.2. *Eine Baumzerlegung $\langle\{X_i \mid i \in I\}, T\rangle$ wird* kanonische *Baumzerlegung genannt, falls die folgenden Bedingungen erfüllt sind:*

1. *Jeder Knoten von T hat höchstens zwei Kinder.*
2. *Wenn ein Knoten i zwei Kinder i und j hat, dann gilt $X_i = X_j = X_k$ (in diesem Fall heißt i Join Node).*
3. *Falls ein Knoten i nur ein Kind j besitzt, muss einer der beiden folgenden Fälle gelten:*
   (a) $|X_i| = |X_j| + 1$ *und* $X_j \subset X_i$ *(in diesem Fall heißt i Insert Node).*
   (b) $|X_i| = |X_j| - 1$ *und* $X_i \subset X_j$ *(in diesem Fall heißt i Forget Node).*

Es ist nicht allzu schwer eine gegebene Baumzerlegung in eine kanonische Baumzerlegung umzuformen. Genauer gesagt, gilt Folgendes (siehe [14, Lemma 13.1.3]).

Proposition 5.3. *Ist eine Baumzerlegung der Weite k auf n Knoten für einen Graphen G gegeben, so kann man eine kanonische Baumzerlegung der Weite k mit $O(n)$ Knoten für G in $O(n)$ Schritten finden.*

Um uns genauer auf die relevanten Substrukturen einer Baumzerlegung beziehen zu können, machen wir folgende Festlegungen: T_i sei der Teilbaum von T der im Knoten i gewurzelt ist. G_i ist dann der Subgraph von G, der durch die Knoten in den Beuteln von T_i induziert wird, also $G_i := G[\cup_{j \in T_i} X_j]$. Im Weiteren benutzen wir Y_i, wenn wir $(\cup_{j \in V(T_i)} X_j) \setminus X_i$ meinen.
Wir werden noch drei Eigenschaften von kanonischen Baumzerlegungen zeigen, von denen wir im Weiteren Gebrauch machen werden.

Beobachtung 5.4.

1. *Für alle $y \in Y_i$ gibt es keinen Beutel $X_k \ni y$, sodass i ein Nachfahre von k in T ist.*
2. *Sei X_l ein Beutel mit Nachfahre X_i in T und $x \in X_l \setminus X_i$.*
   (a) *Es gibt keinen Beutel $X_k \ni x$, so dass k Nachfahre von i in T ist.*
   (b) *Es gibt keine Kante $\{y, x\} \in E$ mit $y \in Y_i$.*
3. *Sei i ein Join Node mit Kindern j und k. Für alle $v_1 \in Y_j$ und $v_2 \in Y_k$ gilt $v_1 \neq v_2$.*

Beweis.

1. Da $y \in Y_i$ gilt, gibt es einen Beutel $X_j \ni y$, welcher Nachfahre von X_i ist. Angenommen es gibt solch ein X_k, dann gilt $X_j \cap X_k \subseteq X_i$ und damit $y \in X_i$. Widerspruch.
2. (a) Angenommen es gibt solch ein X_k, dann gilt $X_k \cap X_l \subseteq X_i$, also $x \in X_i$. Widerspruch.
   (b) Angenommen es gibt diese Kante. Dann gibt es auch einen Beutel X_h mit $\{x, y\} \subseteq X_h$. Nach 1. muss X_h ein Nachfahre von X_i sein. Mit 2.(a) wissen wir aber, dass jeder Nachfahre von X_i nicht x enthält. Widerspruch.
3. Da j Kind von i ist und $v_1 \in Y_j$, so muss es einen Nachfahren $w_1$von i mit $v_1 \in X_{w_1}$ geben. Analog gibt es einen Nachfahren w_2 von i mit $v_2 \in X_{w_2}$. Angenommen $v_1 = v_2$, dann gilt $v_1 \in X_{w_1} \cap X_{w_2}$. Da sowohl w_1 als auch w_2 Nachfahren von i sind, gilt $X_{w_1} \cap X_{w_2} \subseteq X_i$ und es folgt $v_1 \in X_i$. Widerspruch. □

Zur späteren Laufzeitabschätzung brauchen wir noch die folgenden Feststellung.

Lemma 5.5. *Es gilt $k! \in O(2^{k^2})$.*

Beweis. Wir müssen zeigen, dass es eine Konstante c und ein k_0 gibt, so dass $\frac{k!}{2^{k^2}} \leq c$ für alle $k \geq k_0$ gilt. Dies folgt mit $k_0 = 1$, da

$$\frac{k!}{2^{k^2}} = \frac{1}{2^k} \cdot \frac{2}{2^k} \cdot \ldots \cdot \frac{k}{2^k} \leq \underbrace{\frac{k}{2^k}}_{\leq \frac{1}{2}} \cdot \frac{k}{2^k} \cdot \ldots \cdot \frac{k}{2^k} \leq \frac{1}{2}.$$

□

5.2 Dynamisches Programmieren für Graphen beschränkter Baumweite

Im Folgenden wollen wir den bereits erwähnten **FPT**-Algorithmus entwickeln. Als erstes untersuchen wir Gemeinsamkeiten zwischen GSP-Graphen und k-Terminal Graphen. Wir werden sehen, dass der Lösungsansatz für GSP-Graphen auf k-Terminal Graphen direkt übertragbar ist. Für Graphen mit beschränkter Baumweite wird sich zeigen, dass wir noch weitere Modifikationen vornehmen müssen.

5.2.1 GSP-Graphen und k-Terminal Graphen

Eine Graphklasse, die zu GSP-Graphen als auch zu Graphen mit beschränkter Baumweite, eine gewisse Verwandtschaft besitzt, ist die Klasse der k-Terminal Graphen. Ein *Terminal Graph* ist ein Triple $G(V, E, X)$, so dass $G(V, E)$ ein Graph ist und $X \subseteq V$ eine Menge, deren Elemente von 1 bis $|X|$ nummeriert sind. X ist die Terminalmenge von $G(V, E, X)$. Ein Knoten $v \in V$ heißt *Terminal* von $G(V, E, X)$, falls $v \in X$ und *innerer Knoten* von $G(V, E, X)$, falls $v \in V \setminus X$ gilt. Ein Terminal Graph $G(V, E, X)$ ist ein *k-Terminal Graph*, falls $|X| = k$. Damit stellt ein k-Terminal Graph bezüglich der Terminale eine Verallgemeinerung eines GSP-Graphen dar.
Ähnlich wie bei GSP-Graphen gibt es für k-Terminal Graphen eine Operation, die aus zwei gegebenen einen neuen k-Terminal Graphen erstellt. Falls $G_1(V_1, E_1, X_1)$ und $G_2(V_2, E_2, X_2)$ k-Terminal Graphen sind, so ist $G_1 \oplus G_2$ der Graph, den wir durch die disjunkte Vereinigung von $G_1(V_1, E_1)$ und $G_2(V_2, E_2)$, und durch die Identifizierung der j-ten Terminale in X_1 und X_2 für alle $1 \geq j \geq k$ erhalten.
GSP-Graphen sind eine besondere Form von 2-Terminal Graphen. Aus diesem Grund gibt es einen analogen Lösungsansatz für k-Terminal Graphen. In diesem neuen Kontext betrachten wir anstatt zwei Terminalzuständen ein Zustandstupel $(a_1, \ldots, a_k)$ mit $a_i \in \{I, II, III, IV, V\}$. Wir stellen analog zu den GSP-Graphen eine Tabelle für einen k-Terminal Graphen $G(V, E, X)$ auf:

$$T_G(a_1, \ldots, a_k, P_{11}, \ldots, P_{1k}, \ldots, P_{k1}, \ldots, P_{kk})$$

Die Bedeutung, die dahinter steckt, ist geradewegs analog: In jedem Eintrag finden wir die kleinstmögliche Größe einer partiell Gültigen Ausrichtung für G, so dass am j-ten Terminal Zustand a_j herrscht, und, dass für alle $p \in P_{ij}$ ein Pfad der Klasse p vom Knoten i zum Knoten j verläuft. Wenn wir für einen k-Terminal Graphen bezüglich der Operation $\oplus$ einen Parse-Tree besitzen, so können wir eine Gültige Ausrichtung mit kleinstmöglicher Bezugsmenge wie folgt finden: Für jedes Blatt generieren wir die Tabelleneinträge durch vollständige Suche. Danach traversieren wir den Parse-Tree wieder Bottom-Up. Für jeden inneren Knoten des Parse-Trees können wir die Tabelleneinträge mit nur kleinen kanonischen Abwandlungen wie bei den GSP-Graphen aus den Lösungen der Kindern bestimmen. Sind wir bei der Wurzel angelangt, so ist die Lösung, ganz in Korrespondenz zu GSP-Graphen, der kleinstmögliche Eintrag, sodass $a_i \in \{I, II, IV\}$. Die hauptsächliche Komplexität hängt hierbei von der Initialisierung der Blätter durch vollständige Suche ab.
Lässt sich diese Idee für Graphen mit Baumweite k weiterentwickeln? Auch in diesem Fall besitzen wir einen Baum T den wir Bottom-Up traversieren können. Ebenfalls könnte man für die Blätter mit vollständiger Suche alle Lösungen für diese ermitteln. Wir wollen uns aus Gründen der Einfachheit auf kanonische Baumzerlegungen festlegen. Betrachten wir insbesondere einen Join Node X_i mit Kindern X_j und X_k. Man kann sich jetzt figurativ diese Situation so ausmalen, dass X_i dadurch entsteht, dass die Knoten in X_j und X_k miteinander identifiziert werden. Doch da gibt es einen großen Unterschied zu k-Terminal Graphen. Führen wir die Operation $G_1 \oplus G_2$ für zwei k-Terminal Graphen $G_1(V_1, E_1.X_1)$ und $G_2(V_2, E_2, X_2)$ durch, werden nur die entsprechenden Terminale der beiden Graphen miteinander identifiziert. Die Kanten und restlichen Knoten ergeben sich durch disjunkte Vereinigung. Insbesondere gilt $E_1 \cap E_2 = \emptyset$. Im Falle des

Join Node X_i gilt für die Kinder X_j und X_k aber $E(X_j) \cap E(X_k) \neq \emptyset$. Es werden hier also nicht nur Knoten identifiziert, sondern auch Kanten. Dann ist es aber auch schnell einsehbar, dass die Terminalzustände hier ein unzureichendes Konzept darstellen. Sie sagen ja nichts über den Zustand der Kanten aus. Es darf beispielsweise nicht sein, dass für eine Kante $e \in E(X_j)$ bzw. $e \in E(X_k)$ bezogen auf X_j dann $e \in F_u$ und bezogen auf X_k dann $e \in F_b$ gilt. Wir müssen deswegen auch die Ausrichtungen der Kanten als Zustände in Betracht ziehen.

5.2.2 Zustände für die Beutel

Wir wollen nun einen ungerichteten zusammenhängenden Graph $G(V, E)$ mit $V := \{v_1, v_2, \ldots, v_n\}$ und eine kanonischen Baumzerlegung $\langle \{X_i \mid i \in V(T)\}, T\rangle$ für G mit Baumweite k betrachten. Wir werden in unserem Algorithmus den Baum T, ähnlich wie den Parse-Tree eines GSP-Graphen, auf Bottom-Up-Weise traversieren. Während dieses Prozesses werden wir für jeden Beutel X_i die möglichen GA's für G_i berechnen. Dabei müssen gewisse Zustände respektiert werden. Ein Zustand von X_i ist eine Kombination aus Zuständen für alle geordneten Paare von Knoten aus X_i, Zuständen von Knoten in X_i und Zuständen für Kanten aus $G[X_i]$.
Wenn wir im Laufe des Algorithmus einen Beutel X_i bearbeiten, dann gibt es, ähnlich wie bei dem Algorithmus für GSP-Graphen, wieder Knoten in G_i, die wir im weiteren Verlauf nicht mehr betrachten werden. Dies sind genau die Knoten in Y_i. Die Knoten die wir momentan betrachten sind die in X_i. Einige von ihnen werden eventuell auch in nachfolgenden Schritten betrachtet, andere nicht. Für die Knoten in Y_i müssen entweder bereits die Eigenschaften für eine GA vorhanden sein oder wir können sie mittels Knoten aus X_i in späteren Schritten noch sicherstellen.

Um mit einem Dynamic-Programming-Ansatz auf T eine Lösung für G bestimmen zu können, müssen wir für jeden Knoten $i \in T$ mit Beutel X_i eine konstante Menge von relevanten Teillösung berechnen. Es ist nun entscheidend wie die Struktur dieser Teillösung ist. Im Fall eines GSP-Graphen, sahen wir bereits, dass diese Teillösungen 2-PGA's der Subgraphen von G waren, die zu den Knoten im Parse-Tree $PT(G)$ korrespondieren.

Proposition 5.6. *Sei $\langle \{X_i \mid i \in I\}, T\rangle$ eine Baumzerlegung für einen Graphen G. Sei D eine kleinstmögliche GA für G. Ist $D_i := D[T_i]$, so ist D_i eine ℓ-PGA mit $V_\eta = X_i$, $V_\theta = Y_i$ und $|X_i| = \ell$.*

Beweis. Mit Beobachtung 5.4 2b) ist uns bekannt, dass es zwischen $D[Y_i]$ und $D[V(T) \setminus V(T_i)]$ keine Kanten gibt. Ein Knoten $y \in Y_i$ hat dann in D und in $D[Y_i]$ denselben Grad und da D eine GA ist muss $indeg_{F_u}(y) = 1$ in beiden Fällen gelten. Für Knoten $x \in X_i$ kann es in D aber Kanten zu Knoten $u \in D[V(T)\setminus V(T_i)]$ geben. Aus diesem Grund kann es sein, dass die Knoten in X_i in D_i eventuell eine Kante $(u, x) \in F_u$ verloren haben. Deshalb muss $indeg_{F_u}(x) \leq 1$ in D_i gelten. Da D keinen alternierenden Kreis enthält, muss y in D auf einem Observationspfad liegen, der in einen Knoten $u \in A$ entspringt. In D_i ist dieser Observationspfad entweder vollständig enthalten oder unterbrochen und endet in einem Knoten aus X_i. Also ist D_i eine ℓ-PGA wie behauptet. □

Wir werden also für einen Knoten i in T wieder ℓ-partiell Gültige Ausrichtungen erstellen, wobei im Weiteren immer $\ell = |X_i|$ und $V_\eta = X_i$ und $V_\theta = Y_i$

gelten. Da $|X_i| \leq k$ sprechen wir immer von k-partiell Gültigen Ausrichtungen (k-PGA). Gleichsam wie 2-PGA's für GSP-Graphen, werden wir k-PGA's bezüglich ihrer Pfadklassen zwischen zwei Knoten aus X_i und der Zustände der Knoten aus X_i unterscheiden. Zuzüglich müssen wir k-PGA's noch hinsichtlich der Ausrichtung der Kanten aus $E(X_i)$ differenzieren. Aus algorithmischer Sicht bedeutet dies, dass es für jeden Knoten $i \in T$ eine k-PGA für G_i gibt, die Teil der kleinstmöglichen GA für G ist. Wir müssen also für einen Knoten i in T k-PGA's für G_i berechnen, die wir nur durch Knotenzustände, Knotenpaarzustände und Kantenzustände unterscheiden müssen. Wenn wir eine k-PGA $D(V, F)$ haben, so wollen wir im weiteren Verlauf $D[Y_i]$ bezüglich jeweils zweier Knoten $u, v \in X_i$ untersuchen. Um Notation verkürzen zu können, soll für eine Teilmenge $Y \subseteq V$ und zwei Knoten $u, v \in X_i$ $D\langle Y, u, v\rangle := (Y \cup \{u, v\}, F(Y \cup \{u\}) \cup F(Y \cup \{v\}))$ gelten.
Der entscheidende Punkt während der Dynamischen Programmierung ist, dass wir die alternierenden Kreise in k-PGA's D_i für G_i finden. In der induzierten k-PGA für $D_i[X_i]$ können wir mittels vollständiger Suche alle alternierenden Pfade finden. Da X_i höchstens k Knoten besitzt, ist die Zeit die wir hierfür brauchen exponentiell in k beschränkt. Das Finden von alternierenden Kreisen, die teilweise in $D_i\langle Y_i, u, v\rangle$ liegen, für Knoten $u, v \in X_i$ mit $u \neq v$, benötigt Information über die Existenz von alternierenden Pfaden zwischen u und v in $D_i\langle Y_i, u, v\rangle$. Ganz analog zu GSP-Graphen reicht es hierfür aus, sich die Pfadklassen von u nach v bzw. v nach u in $D_i\langle Y_i, u, v\rangle$ zu merken. Von nun an bezeichnet D_i immer eine k-PGA für G_i.

Definition 5.7. *Sei $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ ein Beutel in einer Baumzerlegung $\langle\{X_i \mid i \in V(T)\}, T\rangle$. Ein Knotenpaarzustand für X_i ist eine Funktion $z_{X_i:p} : \{x_{i_1}, \ldots, x_{i_{n_i}}\}^2 \setminus \{(v, v) \mid v \in X_i\} \to \mathcal{P}(\{1, \ldots, 4\})$.*

Ein Knotenpaarzustand $z_{X_i:p}$ gibt alle Pfadklassen zwischen zwei verschiedenen Knoten $u, v \in X_i$ einer k-PGA D_i in $D_i\langle Y_i, u, v\rangle$ wieder. Da $|X_i| \leq k$ gilt und $\{1, \ldots, 4\}$ 16 Teilmengen enthält, so gibt es maximal 16^{k^2} Knotenpaarzustände. Da wir die alternierenden Pfade in $D_i[X_i]$ durch vollständige Suche ermitteln, können wir mit Hilfe dieser Informationen alle alternierenden Kreis finden, deren eine Hälfte in $D_i[X_i]$ und deren andere in $D_i\langle Y_i, u, v\rangle$, mit $u, v \in X_i$, liegt.
Wie eingangs bereits erwähnt, müssen wir auch die Orientierung der Kanten in $E(X_i)$ beachten. Da es für jede Kante in einer Ausrichtung drei Möglichkeiten gibt, definieren wir folgendes:

Definition 5.8. *Sei X_i ein Beutel in einer Baumzerlegung $\langle\{X_i \mid i \in V(T)\}, T\rangle$ mit $E(X_i) = \{e_{i_1}, \ldots, e_{i_{m_i}}\}$ und $e_{i_j} = \{u_{i_j}, v_{i_j}\}$. Ein Kantenzustand ist dann eine Abbildung $z_{X_i:e} : E(X_i) \to \{u_{i_1}v_{i_1}, v_{i_1}u_{i_1}, \ldots, u_{i_{m_i}}v_{i_{m_i}}, v_{i_{m_i}}u_{i_{m_i}}, \perp\}$, wobei für alle e_{i_j} gilt $z_{X_i:e}(e_{i_j}) \in \{u_{i_j}v_{i_j}, v_{i_j}u_{i_j}, \perp\}$.*

Wenn wir nachfolgende Semantik für alle $e_{i_j} \in E(X_i)$ festlegen, so repräsentiert $z_{X_i:e}$ die induzierte k-PGA in $D_i[X_i]$ für eine k-PGA D_i:

- ✯ $z_{X_i:e}(e_{i_j}) = u_{i_j}v_{i_j}$: e_{i_j} ist von u_{i_j} nach v_{i_j} ausgerichtet.
- ✯ $z_{X_i:e}(e_{i_j}) = v_{i_j}u_{i_j}$: e_{i_j} ist von v_{i_j} nach u_{i_j} ausgerichtet.
- ✯ $z_{X_i:e}(e_{i_j}) = \perp$: e_{i_j} ist von u_{i_j} nach v_{i_j} als auch von v_{i_j} nach u_{i_j} ausgerichtet.

Da für einen Beutel X_i immer $|E(X_i)| \leq k^2$ gilt und jede Kante auf drei Arten ausgerichtet werden kann, gibt es maximal 3^{k^2} Kantenzustände für X_i.
Wenn wir k-PGA's durch ihre Kantenzustände unterscheiden und sie als relevantes Merkmal speichern, so können wir auch für alle Knoten $x \in X_i$ die inzidenten Kanten aus $E(D_i[X_i]) \cap F_b$ und $E(D_i[X_i]) \cap F_u$ für eine k-PGA D_i bestimmen. Wir wissen allerdings nichts über die zu x inzidenten Kanten aus $E(D_i[Y_i \cup \{x\}])$, wenn die Knoten in Y_i in keiner Weise als Unterscheidungsmerkmal für k-PGA's dienen sollen. Die Kanten aus $E(D_i[Y_i \cup \{x\}])$ sind im besonderen bei einem Join Node i mit Kindern j und k von Interesse. Denn hier findet für alle Knoten in den Beuteln X_j und X_k eine Knotenidentifikation mit ihrem jeweils korrespondierenden Knoten im anderen Beutel statt. Für einen Knoten x müssen wir Information über $E(D_j[Y_j \cup \{x\}]) \cap F_u$ und $E(D_k[Y_k \cup \{x\}]) \cap F_u$ speichern, damit nach der Knotenidentifikation in G_i keine der geforderten Gradbeschränkungen an einen Knoten in einer k-PGA verletzt ist. Zum Beispiel wenn $(v_1, x) \in E(D_j[Y_j \cup \{x\}]) \cap F_u$ und $(v_2, x) \in E(D_k[Y_k \cup \{x\}]) \cap F_u$ für k-PGA's D_j und D_k für G_j und G_k gilt. Setzen wir $D_j[X_i] = D_k[X_i]$ noch voraus, so können wir eine Ausrichtung D_i für G_i generieren: $D_i(V(G_i), E(D_j) \cup E(D_k))$. Dann würde aber $indeg_{F_u}(x) \geq 2$ gelten. Dieser Umstand kann auch später nicht mehr rückgängig gemacht werden, da die Kanten (v_1, x) und (v_2, x) nie mehr betrachtet werden. Um dies zu verhindern, reicht es, wenn wir folgende fünf Knotenzustände unterscheiden und als Information über $D_i[Y_i]$ speichern.

Definition 5.9. *Sei $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ ein Beutel in einer Baumzerlegung $\langle \{X_i \mid i \in V(T)\}, T \rangle$. Ein Knotenzustand ist dann eine Abbildung $z_{X_i:t} : \{x_{i_1}, \ldots, x_{i_{n_i}}\} \to \{I, \ldots, V\}$ mit folgender Semantik für alle $x_{i_j} \in X_i$ in einer k-PGA für G_i:*

- $z_{X_i:t}(x_{i_j}) = I$ *: Es gibt genau eine Kante $(v_1, x_{i_j}) \in F_u$ mit $v_1 \in Y_i$ und keine Kante $(x_{i_j}, v_2) \in F_u$ mit $v_2 \in Y_i$.*
- $z_{X_i:t}(x_{i_j}) = II$ *: Es gibt genau eine Kante $(v_1, x_{i_j}) \in F_u$ und eine Kante $(x_{i_j}, v_2) \in F_u$ mit $v_1, v_2 \in Y_i$.*
- $z_{X_i:t}(x_{i_j}) = III$ *: Es gibt weder $(v_1, x_{i_j}) \in F_u$ noch $(x_{i_j}, v_2) \in F_u$ mit $v_1, v_2 \in Y_i$.*
- $z_{X_i:t}(x_{i_j}) = IV$ *: x_{i_j} gehört zur Bezugsmenge und es gibt keine Kante $(v_1, x_{i_j}) \in F_u$ mit $v_1 \in Y_i$.*
- $z_{X_i:t}(x_{i_j}) = V$ *: Es gibt genau eine Kante $(x_{i_j}, v_2) \in F_u$ mit $v_2 \in Y_i$ und keine Kante $(v_1, x_{i_j}) \in F_u$ mit $v_1 \in Y_i$.*

Für jeden Knoten $x \in X_i$ gibt es fünf Zustände. Mit $|X_i| \leq k$ gibt es dann für X_i maximal 5^k Knotenzustände.
Die Knoten-, Kanten- und Knotenpaarzustände sind die Informationen, die wir für einen Beutel und dessen k-PGA's speichern müssen. Deshalb fassen wir diese drei Aspekte zusammen.

Definition 5.10. *Sei $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ mit $E(X_i) = \{e_{i_1}, \ldots, e_{i_{m_i}}\}$ ein Beutel in einer Baumzerlegung $\langle \{X_i \mid i \in V(T)\}, T \rangle$. Ein Zustand für einen Beutel X_i ist dann eine Abbildung*

$$z : \{x_{i_1}, \ldots, x_{i_{n_i}}\}^2 \setminus \{(v, v) \mid v \in X_i\} \cup E(X_i) \cup X_i \to \mathcal{P}(\{1, \ldots, 4\}) \cup \{u_{i_1}v_{i_1}, v_{i_1}u_{i_1}, \ldots, u_{i_{m_i}}v_{i_{m_i}}, v_{i_{m_i}}u_{i_{m_i}}, \perp\} \cup \{I, \ldots, V\}$$

wobei gilt

$$z_{X_i}(x) = \begin{cases} z_{X_i:p}(x) : x \in \{x_{i_1}, \ldots, x_{i_{n_i}}\}^2 \setminus \{(v,v) \mid v \in X_i\} \\ z_{X_i:e}(x) : x \in E(X_i) \\ z_{X_i:t}(x) : x \in X_i \end{cases}$$

Mit dem Wissen über die einzelnen Anzahlen der Kanten-, Knoten- und Knotenpaarzustände lässt sich folgern, dass es maximal $16^{k^2} \cdot 3^{k^2} \cdot 5^k$ Beutelzustände für einen Beutel X_i gibt.
Wenn wir im Folgenden davon sprechen, dass eine k-PGA D_i für G_i einen Zustand z_{X_i} respektiert, dann bedeutet dies, dass D_i die zu z_{X_i} entsprechende Kantenausrichtung, die entsprechenden Pfadklassen zwischen zwei Knoten aus X_i und die entsprechenden Knotenzustände bezüglich X_i aufweist. Bemerkenswert ist auch, dass jede k-PGA für G_i einen Beutelzustand z_{X_i} für X_i festlegt. Wichtig ist ebenfalls, dass wir k-PGA's für G_i nur anhand des Zustandes, den sie respektieren, unterscheiden wollen. Denn ist D_i^1 eine k-PGA für G_i, die Zustand z_{X_i} respektiert, und Teil der kleinstmöglichen Lösung für G ist, so kann D_i^1 durch jede weitere k-PGA D_i^2, die z_{X_i} respektiert und eine gleich große Bezugsmenge hat, substituiert werden. Das Resultat der Substitution ist auch eine kleinstmögliche GA für G. Wenn wir nun für jeden Beutel X_i für alle möglichen Beutelzustände z_{X_i} die kleinstmögliche k-PGA für G_i, die z_{X_i} respektiert, bestimmen, so halten wir die Bausteine für die Lösung von GAKB für G in den Händen.

5.2.3 Aufbau und Initialisierung der Tabelle

Ein Zustand z_{X_i} beinhaltet also jegliche Information mit Hilfe derer wir entscheiden können, inwiefern eine k-PGA D_i für G_i, die z_{X_i} respektiert, in einer Lösung für G verwendet werden kann. Jede weitere k-PGA D_i' für G_i, die ebenfalls z_{X_i} respektiert und deren Bezugsmengengröße mit der von D_i übereinstimmt, kann D_i in einer Lösung für G substituieren.
Wir stellen also eine Tabelle T_{X_i} für jeden Beutel X_i in einer Baumzerlegung T auf, die mit einem Beutelzustand z_{X_i} indexiert wird. Dabei soll für einen Eintrag Folgendes gelten:

$$T_{X_i}(z_{X_i}) = \min_{A \subseteq V(G_i)} \{|A| : \exists \text{ k-PGA bezüglich } A \text{ für } G_i, \text{ die } z_{X_i} \text{ respektiert}\}$$

Da wir einen Bottom-Up-Ansatz auf T verfolgen, müssen wir zunächst die Initialisierung der Blätter von T klären. Richten wir nun unser Augenmerk auf einen Beutel X_i der ein Blatt in T ist. Für X_i gibt es $16^{k^2} \cdot 3^{k^2} \cdot 5^k$ mögliche Zustände. Doch für viele unter diesen Zuständen lässt sich keine k-PGA für X_i finden, welche den entsprechenden Zustand respektiert. Da X_i ein Blatt ist, gilt $Y_i = \emptyset$. Somit gibt es auch keine Kanten zwischen X_i und Y_i. Also gibt es für alle Beutelzustände z_{X_i}, für die es ein $x \in X_i$ mit $z_{X_i}(x) \neq III$ und $z_{X_i}(x) \neq IV$ gibt, keine k-PGA, welche z_{X_i} respektiert. Ebenfalls folgt dann für $u, v \in X_i$ und jede k-PGA D_i, dass $D_i \langle Y_i, u, v\rangle = (\{u, v\}, \emptyset)$ gilt. Somit kann es keine alternierenden Pfade zwischen u und v in $D_i \langle Y_i, u, v\rangle$ geben. Deswegen existiert für jeden Beutelzustand z_{X_i}, für den es Knoten $u, v \in X_j$ mit $z_{X_i}(u, v) \neq \emptyset$ gibt, auch keine k-PGA, die z_{X_i} respektiert.
Jeder Zustand legt auch eine Ausrichtung der Kanten in $E(X_i)$ fest. Seien $F(X_i)$ diese Kanten. Damit es eine k-PGA gibt, die z_{X_i} respektiert, muss für jeden

Knoten $x \in X_i$ mit $z_{X_i}(x) = III$ in dieser Ausrichtung $indeg_{F_u \cap F(X_i)}(x) \leq 1$ und $outdeg_{F_u \cap F(X_i)}(x) \leq 1$ und für Knoten $x' \in X_i$ mit $z_{X_i}(x') = IV$ $indeg_{F_u \cap F(X_i)}(x) = 0$ innerhalb von $G[X_i]$ gelten. Ebenso dürfen keine alternierenden Kreise in G_i vorkommen. Ersteres können wir dadurch ermitteln, dass wir die Nachbarschaft jedes Knotens betrachten. Dies benötigt nur $O(k^2)$ Schritte. Die alternierenden Kreise in G_i detektieren wir, indem wir alle in G_i enthaltenen Kreise explizit aufzählen. Wir können die alternierenden Kreise finden, indem wir alle geordneten Teilmenge aus Knoten in X_i betrachten. Ist $S = (x_{s_1}, \ldots, x_{s_l})$ solch eine geordnete Menge, dann stellt $x_{s_1}, \ldots, x_{s_l}, x_{s_1}$ einen Kreis dar. Umgekehrt ist auch jeder Kreis in G_i durch solch eine geordnete Menge beschreibbar. Wenn wir alle diese Mengen betrachten und die korrespondierenden Kreise darauf testen, ob sie alternierend sind, so finden wir auch alle in $G[X_i]$ enthaltenen alternierenden Kreise. Es gibt nicht mehr als $\sum_{i=0}^{k} \binom{k}{i}$ ungeordnete Teilmengen von X_i. Jede dieser ungeordneten Teilmengen mit Größe i steht für $i!$ geordnete Teilmengen. Insgesamt müssen wir dann $\sum_{i=0}^{k} \binom{k}{i} \cdot i!$ viele geordnete Teilmengen von X_i inspizieren. Da aber gilt

$$\sum_{i=0}^{k} \binom{k}{i} \cdot i! = \sum_{i=0}^{k} \frac{k!}{i!(k-i)!} \cdot i! = k! \sum_{i=0}^{k} \frac{1}{i!} \leq k! \sum_{i=0}^{k} \frac{1}{2^i} \leq 2k!.$$

kann dies pro Zustand in $O(k!)$ Zeit geschehen. Da es $16^{k^2} \cdot 3^{k^2} \cdot 5^k$ Beutelzustände z_{X_i} gibt, können wir also festhalten, dass ein Blatt in Zeit $O(16^{k^2} \cdot 3^{k^2} \cdot 5^k \cdot k!)$ initialisiert werden kann. Weiter hat eine kanonische Baumzerlegung $O(n)$ Beutel und deshalb geschieht die Initialisierung aller Beutel in $O(16^{k^2} \cdot 3^{k^2} \cdot 5^k \cdot k! \cdot n)$ Schritten. Im Weiteren verwenden wir die Notation $valid(G[X_i], z_{X_i})$, um eine Prozedur zu bezeichnen, die entscheidet ob die Kantenausrichtung, die z_{X_i} festlegt, eine gültige k-PGA mit $V_\eta = X_i$ und $V_\theta = \emptyset$ für $G[X_i]$ darstellt. Für jeden Zustand z_{X_i} setzen wir dann $T_{X_i}(z_{X_i}) := \infty$ falls gilt

$$(\exists x \in X_i : z_{X_i}(x) \notin \{III, IV\}) \vee (\exists u, v \in X_i : z_{X_i}(u, v) \neq \emptyset)$$
$$\vee valid(G[X_i], z_{X_i}) = \text{`}false\text{`}).$$

Für alle Zustände, die dies nicht erfüllen, müssen wir die Knoten zählen, die in der Bezugsmenge sind, um den korrekten Tabelleneintrag zu generieren:

$$T_{X_i}(z_{X_i}) = |\{x \mid x \in X_i,\ z_{X_i}(x) = IV\}|$$

Wir können also folgendes festhalten:

Lemma 5.11. *Die Initialisierung der Tabelleneinträge der Blätter von T benötigt $O(16^{k^2} \cdot 3^{k^2} \cdot 5^k k! \cdot n)$ Schritte.*

5.2.4 Aktualisierungsprozess

Nach der Initialisierung der Blätter besuchen wir die Beutel der Baumzerlegung auf Bottom-Up-Weise, bei den Blättern beginnend bis zur Wurzel. In jedem dieser Schritte müssen wir die entsprechenden Tabelleneinträge für einen Beutel X_i generieren. Wie wir dies tun ist davon abhängig ob X_i ein Join, Insert oder Forget Node ist. Generell müssen wir für jeden Beutelzustand z_{X_i} für X_i Folgendes tun:

- ✯ Überprüfe, ob die durch z_{X_i} induzierte Kantenausrichtung eine k-PGA von $G[X_i]$ darstellt.

- ✯ Berechne die Menge $Z^{z_{X_i}}$ des 'passenden' Beutelzustandes (bzw. Beutelzustandspaare) des Kindbeutels X_j (bzw. der Kindbeutel X_j und X_k). Die formale Definition von passend müssen wir noch jeweils für Join, Insert und Forget Nodes angeben.
- ✯ Überprüfe hierfür jeden passenden Beutelzustand $z_{X_j} \in Z^{z_{X_i}}$ (bzw. jedes Beutelzustandspaar), ob eine k-PGA für G_j, die z_{X_j} respektiert, eine k-PGA für G_i ist, die z_{X_i} respektiert.
- ✯ Basierend auf den hierdurch ermittelten Tabelleneinträgen der Beutelzustände (bzw. Beutelzustandspaaren) in T_{X_j}, berechne die Einträge für T_{X_i}.

Wir werden für Join, Insert und Forget Nodes angeben wie, wir dies erreichen wollen.

Insert Nodes

Sei also i ein Insert Node mit Kindknoten j. O.b.d.A. ist es uns möglich $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}, x\}$ und $X_j = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ vorauszusetzen. Wir wollen nun $Z^{z_{X_i}}$, die Menge der 'passenden' Zustände, für jeden Zustand z_{X_i} für X_i ermitteln. Mit Beobachtung 5.4 wissen wir, dass zwischen dem Knoten x und jedem Knoten aus Y_j keine Kante in G besteht. Da $Y_j = Y_i$ gilt, folgt selbiges auch für Y_i. Haben wir einen Zustand z_{X_i} und es gilt $z_{X_i}(x) \neq III$ und $z_{X_i}(x) \neq IV$ oder gibt es ein $v \in X_j$ mit $z_{X_i}(v, x) \neq \emptyset$ oder $z_{X_i}(x, v) \neq \emptyset$, dann ist dies nicht zu rechtfertigen. Denn damit dies gilt, muss es mindestens eine Kante zwischen x und einem Knoten in Y_i geben. Aus diesem Grund setzen wir für Zustände z_{X_i}, die

$$\begin{aligned}(valid(G[X_i], z_{X_i}) = \text{`}false\text{`}) \vee (z_{X_i}(x) \notin \{III, IV\})\\ \vee(\exists v \in X_j : z_{X_i}(x, v) \neq \emptyset \vee z_{X_i}(v, x) \neq \emptyset)\end{aligned}$$

erfüllen, $Z^{z_{X_i}} := \emptyset$.
Jede k-PGA für G_i ist beschränkt auf G_j ebenfalls eine k-PGA. Wenn es also für einen Zustand z_{X_i} eine k-PGA auf G_i gibt, die z_{X_i} respektiert, so gibt es auch mindestens einen Zustand z_{X_j} für G_j, der mit z_{X_i} auf G_j eingeschränkt übereinstimmt und die entsprechende k-PGA für G_j, die z_{X_j} respektiert. Aus diesem Grund definieren wir für alle Zustände z_{X_i} mit

$$\begin{aligned}(valid(G[X_i], z_{X_i}) = \text{`}true\text{`}) \wedge (z_{X_i}(x) \in \{III, IV\})\\ \wedge(\forall v \in X_j : z_{X_i}(v, x) = \emptyset, z_{X_i}(x, v) = \emptyset),\end{aligned}$$

$$\begin{aligned}Z^{z_{X_i}} := \{z_{X_j} \mid \forall e \in E(G_j) : z_{X_i}(e) = z_{X_j}(e),\ \forall v \in X_j : z_{X_i}(v) = z_{X_j}(v)\\ \forall u, v \in X_j : z_{X_i}(u, v) = z_{X_j}(u, v)\}\end{aligned} .$$

Jeder Zustand z_{X_i} induziert auf G_i eine Ausrichtung R_i. Falls R_i auf G_j eine k-PGA D_j ist, ist die schwierigste Aufgabe nun herauszufinden, ob R_i auf G_i immer noch eine k-PGA ist, die z_{X_i} respektiert. Hierzu müssen wir die Kantenzustände der Kanten, die zu x in $G[X_i]$ inzident sind, untersuchen. Die Anforderungen an eine k-PGA können entweder durch einen Nachbarn von x oder durch einen alternierenden Kreis, der x enthält, verletzt werden. Wenn R_i

aufgrund eines Nachbarknotens v von x keine k-PGA für G_i ist, dann weil v eine der Gradrestriktionen nicht einhält. Beispielsweise, wenn $z_{X_j}(v) = II$ gilt und es eine Kante $(x, v) \in E(R_i) \cap F_u$ gibt. Dann folgt $indeg_{E(R_i)\cap F_u}(v) \geq 2$. Indem wir die inzidenten Kanten in $E(G[X_i])$ und den Knotenzustand jedes Nachbarknotens v von x anschauen, können wir feststellten wieviele Kanten aus $F_u \cap E(R_i)$ inzident zu v sind. Damit können wir überprüfen, ob hierdurch eine Verletzung statt gefunden hat. Genauer gesagt, muss für jeden Knoten $v \in X_i$ folgendes gelten:

- ✯ Falls $z_{X_i}(v) = IV$, dann muss $indeg_{F_u\cap E(R_i)}(v) = 0$ gelten.
- ✯ Falls $z_{X_i}(v) \neq IV$, dann muss $indeg_{F_u\cap E(R_i)}(v) \leq 1$ und $outdeg_{F_u\cap E(R_i)}(v) \leq 1$ gelten.

Um dies zu garantieren, müssen wir nur die Knoten, die in $G[X_i]$ die Nachbarn von x sind, betrachten. Falls dies für einen Knoten v erfüllt ist, notieren wir dies mit $dres(v) = ❀$.

Wenn $valid(G[X_i], z_{X_i}) =$ '$true$' gilt, kann es innerhalb von $R_i[X_i]$ keinen alternierenden Kreis geben. Ein alternierender Kreis in $R_i[\cup_{c\in V(T_i)} X_c]$ kann nur dadurch zustande kommen, dass er durch einen alternierenden Pfad P in $R_i[X_i]$, zwischen Knoten $u, v \in X_j$ verläuft, und einem alternierenden Pfad in $D\langle Y_i, u, v\rangle$ gebildet wird, der ebenfalls zwischen u und v verläuft. P muss den Knoten x enthalten, denn sonst ist P auch in $R_i[\cup_{c\in V(T_j)} X_c]$ enthalten und derselbe alternierende Kreis entsteht. R_i ist aber auf G_j eine k-PGA und kann deshalb diesen alternierenden Kreis nicht enthalten. Wir müssen also noch für alle Knoten $u, v \in X_j$ alle alternierenden Pfadklassen in $R[X_i]$ zwischen u und v ermitteln, die x enthalten. Indem wir wieder alle geordneten Teilmengen von X_i betrachten, können wir diese Pfade durch vollständige Suche identifizieren. Denn wenn $(x_{s_1}, \ldots, x_{s_l})$ solche eine Menge ist, dann ist $(x_{s_1}, \ldots, x_{s_l})$ ein potentieller alternierender Pfad in $R_i[X_i]$. Die vollständige Suche ist dann wieder in $O(k!)$ Schritten zu bewerkstelligen. Auf die alternierenden Pfadklassen in $R_i[X_i]$ zwischen Knoten $u, v \in X_j$, die x enthalten, beziehen wir uns mit

$$p_x : \{x_{i_1}, \ldots, x_{i_{n_i}}\}^2 \setminus \{(v, v) \mid v \in X_j\} \to \mathcal{P}(\{1, \ldots, 4\}).$$

Hiermit sind wir dann auch in der Lage, die Menge der Zustände z_{X_j} von G_j zu bestimmen, die ebenfalls auf G_i eine k-PGA besitzen:

$$L^{z_{X_i}} = \{z_{X_j} \in Z^{z_{X_i}} \mid \forall v \in X_i : dres(v) = ❀,\ \forall u, v \in X_j : p_x(u, v) \oplus_P z_{X_j}(v, u) \neq \dagger\}.$$

$L^{z_{X_i}}$ enthält alle z_{X_j} für G_j, die auf G_j identisch zu z_{X_i} sind, falls es auf G_i eine k-PGA gibt, die z_{X_i} respektiert. Hiermit sind wir nun in der Lage die Einträge für $T_{X_i}(z_{X_i})$ für alle Zustände z_{X_i} für G_i zu berechnen:

$$T_{X_i}(z_{X_i}) = \min_{z_{X_j} \in L^{z_{X_i}}} (T_{X_j}(z_{X_j}) + f_{z_{X_i}}(x))$$

mit

$$f_{z_{X_i}}(x) = \begin{cases} 1 : z_{X_i}(x) = IV \\ 0 : sonst \end{cases}$$

Die Funktion $f_{z_{X_i}}$ gibt gerade an, ob x in der Bezugsmenge ist oder nicht. Zu beachten ist noch, falls $L^{z_{X_i}} = \emptyset$ gilt, dass wir $T_{X_i}(z_{X_i}) := \infty$ setzen.

Lemma 5.12. *Um die Tabelleneinträge für einen Insert Node X_i mit Kind X_j zu generieren brauchen wir $O((16^{k^2} \cdot 3^{k^2} \cdot 5^k)^2 \cdot k^2)$ Schritte.*

Beweis. Für jeden Beutelzustand z_{X_i} für X_i müssen wir $valid(G[X_i], z_{X_i})$ aufrufen und die Funktion p_x berechnen, welches zusammen in $O(k!)$ Schritten zu bewerkstelligen ist. In $O(k^2)$ Schritten ermitteln wir *dres*. Die beiden genannten Aufgaben lassen sich dann für alle Beutelzustände z_{X_i} in $O(16^{k^2} \cdot 3^{k^2} \cdot 5^2 \cdot k!)$ Schritten erledigen. Da gilt $|Z^{z_{X_i}}| \leq 16^{k^2} \cdot 3^{k^2} \cdot 5^k$ und wir für jeden Zustand z_{X_i} $O(k^2)$ Schritte benötigen um zu entscheiden ob z_{X_j} zu $Z^{z_{X_i}}$ gehört, ist der Aufwand für all z_{X_i} in $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2 \cdot k^2)$. Ebenso gilt $|\cup_{z_{X_i}} L^{z_{X_i}}| \leq (16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2$. Um die Einträge für T_{X_i} zu berechnen, müssen wir für jedes z_{X_i} alle Elemente in $L^{z_{X_i}}$ betrachten. Dies benötigt dann eine Zeitkomplexität von $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2)$. Mit Lemma 5.5 wissen wir, dass $16^{k^2} \cdot 3^{k^2} \cdot 5^2 \cdot k! \in O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2 \cdot k^2)$, womit die Behauptung folgt. □

Forget Nodes

Sei i nun ein Forget Node mit Kind j. O.B.d.A. gilt $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ und $X_j = \{x_{i_1}, \ldots, x_{i_{n_i}}, x\}$. Für jeden Zustand berechnen wir wieder die Menge $Z^{z_{X_i}}$ der passenden Beutelzustände z_{X_j} von X_j. Dies bezüglich sagen wir, dass z_{X_j} passend zu z_{X_i} bezüglich eines Knotens $v \in X_i$ ist, falls einer der folgenden Punkte gilt:

1. $z_{X_i}(v) = z_{X_j}(v) \wedge ((\{x, v\} \notin E) \vee (\{x, v\} \in E \wedge z_{X_j}(\{x, v\}) = \bot))$
2. $z_{X_i}(v) = I \wedge z_{X_j}(v) = III \wedge (\{x, v\} \in E \wedge z_{X_j}(\{x, v\}) = xv)$
3. $z_{X_i}(v) = II \wedge z_{X_j}(v) = I \wedge (\{x, v\} \in E \wedge z_{X_j}(\{x, v\}) = vx)$
4. $z_{X_i}(v) = II \wedge z_{X_j}(v) = V \wedge (\{x, v\} \in E \wedge z_{X_j}(\{x, v\}) = xv)$
5. $z_{X_i}(v) = IV \wedge z_{X_j}(v) = IV \wedge (\{x, v\} \in E \wedge z_{X_j}(\{x, v\}) = vx)$
6. $z_{X_i}(v) = V \wedge z_{X_j}(v) = III \wedge (\{x, v\} \in E \wedge z_{X_j}(\{x, v\}) = vx)$

Eine Bedeutung lässt sich hieraus ableiten, wenn man bedenkt, dass $x \in Y_i$ gilt. Das heißt, dass für jeden Knoten $v \in X_j$ für den x ein Nachbar in $G[X_j]$ ist, x kein Nachbar in $G[X_i]$ ist. Das kann zur Folge haben, dass v zu einer weiteren Kante aus F_u inzident ist, so dass als Endknoten x in Y_i zu finden ist. Die möglichen Kanten hier sind (x, v) und (v, x). Dementsprechend ändert sich auch der Knotenzustand für v in X_i. Ist die Kante $\{x, v\}$ jedoch ungerichtet in z_{X_j}, so bleiben die Zustände gleich, da zu v keine, im Vergleich zu X_j, weitere Kante aus F_u mit Endknoten in Y_i inzident ist. In Punkt 3 beispielsweise gilt $z_{X_j}(v) = I$. Wenn es eine Kante $\{x, v\}$ gibt, so verbindet sie bezüglich X_i den Knoten v mit einem Knoten aus Y_i. Bezogen auf X_j liegt diese Kante aber noch in $E(X_j)$. Wenn nun $z_{X_j}(\{x, v\}) = vx$ für eine existente Kante $\{x, v\}$ gilt, so bedeutet dies, dass in X_i im Vergleich zu X_j eine weitere gerichtete Kante zwischen X_i und Y_i existiert. Da $z_{X_j}(v) = I$ gilt, gibt es in X_j bereits eine gerichtete Kante (v', v) mit $v' \in Y_j$. Da auch $v' \in Y_i$ gilt, muss dann $z_{X_i}(v) = II$ folgen. Analoge Überlegungen führen zu den Ausdrücken, die in den weiteren Punkten zu finden sind.

Wenn für alle $v \in X_i$ der Zustand z_{X_j} passend zu z_{X_i} bezüglich v ist, so wollen wir $z_{X_i} \sim z_{X_j}$ schreiben.

Hiermit haben wir festgelegt, welche Zustände z_{X_j} bezüglich der Knotenzustände zu z_{X_i} passen. Wir müssen nun Überlegungen anstrengen, was passend bezüglich der Knotenpaarzustände bedeutet. Klar ist wegen $Y_i = Y_j \cup \{x\}$, dass für $u, v \in X_i$ immer $z_{X_j}(u, v) \subseteq z_{X_i}(u, v)$ gilt. Da aber nun $x \in Y_i$ gilt, ergibt die S1-Konkatenation einer Pfadklasse aus $z_{X_j}(u, x)$ mit einer aus $z_{X_j}(x, v)$ eventuell eine weitere Pfadklasse zwischen u und v in $D_i\langle Y_i, u, v\rangle$, wenn D_i eine k-PGA ist die z_{X_i} respektiert. Ebenfalls gilt nun für jede Kante $\{v, x\}$ mit $v \in X_i$, dass sie ein alternierender Pfad zwischen v und x in $D_i\langle Y_i, u, v\rangle$ ist. Für zwei Nachbarn u, v von x in $G[X_j]$ ist es nun möglich, dass sie in $D_i\langle Y_i, u, v\rangle$ durch einen alternierenden Pfad mit zwei Kanten, der x enthält, verbunden sind. In dem wir in $G[X_j]$ wieder durch vollständige Suche alle alternierenden Pfade aufzählen, die zwischen zwei Knoten $u, v \in X_j$ verlaufen, können wir auch diese in $O(k!)$ Schritten ermitteln. Mit

$$p_a : \{x_{i_1}, \ldots, x_{i_{n_i}}, x\}^2 \setminus \{(v, v) \mid v \in X_j\} \to \mathcal{P}(\{1, \ldots, 4\})$$

nehmen wir auf die Pfadklassen zwischen $u, v \in X_j$ Bezug, so dass die korrespondierenden Pfade nur Kanten aus $E(X_j) \setminus E(X_i)$ besitzen und x enthalten. Wir müssen auch diese Pfade in Betracht ziehen, da sie in $D_i\langle Y_i, u, v\rangle$ verlaufen und nicht in $D_i\langle Y_j, u, v\rangle$.

Bezüglich eines Knotenpaares (u, v) ist der Zustand z_{X_j} zu z_{X_i} dann passend, falls gilt

$$\begin{aligned} z_{X_i}(u, v) =& z_{X_j}(u, v) \cup (z_{X_j}(u, x) \triangle_{S1} z_{X_j}(x, v)) \cup p_a(u, v) \\ & \cup (z_{X_j}(u, x) \triangle_{S1} p_a(x, v)) \cup (p_a(u, x) \triangle_{S1} z_{X_j}(x, v)). \end{aligned}$$

Wenn bezüglich aller Knotenpaare $u, v \in X_i$ ein Zustand z_{X_j} passend zu z_{X_i} ist, so schreiben wir $z_{X_i} \leftrightarrow z_{X_j}$.

Hiermit können wir die Menge $Z^{z_{X_i}}$ aller zu z_{X_i} passenden Zustände z_{X_j} erstellen:

$$Z^{z_{X_i}} = \{z_{X_j} \mid \forall e \in E(X_i) : z_{X_i}(e) = z_{X_j}(e),\ z_{X_i} \sim z_{X_j},\ z_{X_i} \leftrightarrow z_{X_i}\}.$$

Da $x \in Y_i$ gilt, wird x im weiteren Verlauf nach Beobachtung 5.4.1 nie wieder betrachtet werden. Aus diesem Grund muss garantiert sein, dass die Eigenschaften einer k-PGA nicht durch x verletzt werden. Da nun $x \in V_\theta$ gilt, muss der Knoten x also entweder in der Bezugsmenge sein oder aber es muss $indeg_{F_u \cap E(R_i)}(x) = 1$ gelten. Damit können wir die Menge der Zustände $L^{z_{X_i}}$ aufstellen, die Lösungen darstellen:

$$L^{z_{X_i}} = \{z_{X_j} \in Z^{z_{X_i}} \mid z_{X_j}(x) = IV \text{ oder } indeg_{F_u \cap E(R_i)}(x) = 1\}.$$

Für alle $z_{X_j} \in L^{z_{X_i}}$ gibt es eine k-PGA auf G_j, die z_{X_j} respektiert, und auf G_i den Zustand z_{X_i} respektiert. Wenn wir diese Menge haben, können wir sofort die Einträge für T_{X_i} berechnen:

$$T_{X_i}(z_{X_i}) = \min_{z_{X_j} \in L^{z_{X_i}}} (T_{X_j}(z_{X_j})).$$

Weiterhin zu beachten bleibt, dass wir $T_{X_i}(z_{X_i}) := \infty$ setzen, falls $L^{z_{X_i}} = \emptyset$ gilt.

Lemma 5.13. *Um die Tabelleneinträge für einen Forget Node X_i zu generieren brauchen wir $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2 \cdot k^2)$ Zeit.*

Beweis. Um die Abbildung p_a für alle Beutelzustände z_{X_i} aufzustellen, sind $O(16^{k^2} \cdot 3^{k^2} \cdot 5^2 \cdot k!)$ Schritte nötig. Um für einen Zustand z_{X_i} die Menge $Z^{z_{X_i}}$ zu berechnen, brauchen wir $O(16^{k^2} \cdot 3^{k^2} \cdot 5^2 \cdot k^2)$ Zeit, für alle z_{X_i} also $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2 k^2)$ Zeit. Um $L^{z_{X_i}}$ aufzustellen, brauchenn wir $O(16^{k^2} \cdot 3^{k^2} \cdot 5^2 \cdot k)$ und um die Tabelleneinträge für T_{X_i} zu konstruieren $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2)$ Schritte. Insgesamt ergibt sich also eine Zeitkomplexität von $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2 \cdot k^2)$. □

Join Nodes

Sei i nun ein Join Node mit Kindern j und k, sodass $X_i = \{x_{i_1}, \ldots, x_{i_{n_i}}\}$ und $X_i = X_j = X_k$ gilt. Die Betrachtungen, die wir in diesem Fall vornehmen werden, unterscheiden sich nicht allzu sehr von denen für Insert oder Forget Nodes. Ein merklicher Unterschied hier ist aber, dass wir dieses Mal passende Zustandspaare (z_{X_j}, z_{X_k}) für z_{X_i} suchen. Passend ist in erster Instanz ein Paar (z_{X_j}, z_{X_k}), wenn für alle $e \in E(X_i)$ $z_{X_i}(e) = z_{X_j}(e) = z_{X_k}(e)$ gilt. Besitzen wir zwei k-PGA's D_j, D_k, die jeweils z_{X_j} und z_{X_k} respektieren, und z_{X_j} und z_{X_k} 'passen' zueinander, so erhalten wir, wenn wir die Knoten und Kanten in $G[X_j]$ und $G[X_k]$ identifizieren, eine Ausrichtung R_i für G_i. Welche Kriterien müssen erfüllt sein, dass R_i ebenfalls eine k-PGA ist? Innerhalb von $G[X_i]$ können die Gradbedingungen nicht verletzt werden, doch für Kanten, die nach Y_i führen, ist dies möglich. Gilt z.B. für einen Knoten $v \in X_i$ $z_{X_j}(v) = I$ und $z_{X_k}(v) = I$, so gibt es Knoten $u_1 \in Y_j$ und $u_2 \in Y_k$ mit $(u_1, v), (u_2, v) \in F_u$. Mit Beobachtung 5.4.3 folgt $u_1 \neq u_2$. Dann gilt aber $indeg_{F_u \cap E(R_i)}(v) \geq 2$, welches verhindert, dass R_i eine k-PGA ist.

In Anlehnung an diesen Fall sagen wir, dass (z_{X_j}, z_{X_k}) bezüglich eines Knotens $v \in X_i$ zu z_{X_i} passt, falls einer der folgenden Punkte erfüllt ist:

1. $z_{X_i}(v) = I \wedge ((z_{X_j}(v) = I \wedge z_{X_k}(v) = III) \vee (z_{X_j}(v) = III \wedge z_{X_k}(v) = I))$
2. $z_{X_i}(v) = II \wedge ((z_{X_j}(v) = II \wedge z_{X_k}(v) = III) \vee (z_{X_j}(v) = III \wedge z_{X_k}(v) = II)$
   $\vee\ (z_{X_j}(v) = I \wedge z_{X_k}(v) = V) \vee (z_{X_j}(v) = V \wedge z_{X_k}(v) = I))$
3. $z_{X_i}(v) = III \wedge z_{X_j}(v) = III \wedge z_{X_k}(v) = III$
4. $z_{X_i}(v) = IV \wedge z_{X_j}(v) = IV \wedge z_{X_k}(v) = IV$
5. $z_{X_i}(v) = V \wedge ((z_{X_j}(v) = V \wedge z_{X_k}(v) = III) \vee (z_{X_j}(v) = III \wedge z_{X_k}(v) = V))$

Wenn für alle $v \in X_i$ gilt, dass (z_{X_j}, z_{X_k}) passend zu z_{X_i} ist, so schreiben wir in diesem Fall $(z_{X_j}, z_{X_k}) \sim z_{X_i}$.

Weiter sagen wir, dass (z_{X_j}, z_{X_k}) passend zu z_{X_i} bezüglich eines geordneten Paares (u, v) mit $u, v \in X_i$ ist, wenn $z_{X_i}(u, v) = z_{X_j}(u, v) \cup z_{X_k}(u, v)$ der Fall ist. Wenn für alle Paare (u, v) (z_{X_j}, z_{X_k}) zu z_{X_i} passend ist, so schreiben wir $(z_{X_j}, z_{X_k}) \leftrightarrow z_{X_i}$. Dies macht Sinn, denn wenn R_i eine k-PGA sein soll, dann muss $(z_{X_j}, z_{X_k}) \leftrightarrow z_{X_i}$ gelten, da ja R_i gerade aus D_j und D_k besteht.

Damit legen wir die Menge $Z^{z_{X_i}}$ der passenden Zustandspaare fest

$$Z^{z_{X_i}} = \{(z_{X_i}, z_{X_k}) \mid \forall e \in E(X_i) : z_{X_i}(e) = z_{X_j}(e) = z_{X_k}(e),\ (z_{X_j}, z_{X_k}) \sim z_{X_i},\ (z_{X_j}, z_{X_k}) \leftrightarrow z_{X_i}\}$$

Bisher haben wir noch nicht das Entstehen von alternierenden Kreisen in R_i in Augenschein genommen. Wir müssen uns fragen wie solche Kreise entstehen können. In $G[X_i]$ kann kein alternierender Kreis vorkommen, denn sonst gäbe es ihn auch in $G[X_j]$ und $G[X_k]$. Ein alternierender Kreis kann nur entstehen, wenn ein alternierender Pfad mit Endknoten $u_1, v_1 \in X_i$, der in $D_j\langle Y_j, u_1, v_1\rangle$ verläuft, mit einem weiteren zwischen $u_2, v_2 \in X_i$, der in $D_k\langle Y_k, u_2, v_2\rangle$ verläuft, verbunden wird. Dies geschieht wenn es in $G[X_i]$ entsprechende alternierende Pfad zwischen u_1, u_2 und v_1, v_2 gibt. Die alternierenden Pfadklassen zwischen zwei Knoten $u, v \in X_i$, die nur in $R_i[X_i]$ verlaufen, ermitteln wir wieder mit Hilfe der geordneten Teilmengen aus X_i durch vollständige Suche in $O(k!)$ Schritten. Auf diese Pfadklassen zwischen zwei Knoten aus X_i verweisen wir mit der Abbildung

$$p : \{x_{i_1}, \ldots, x_{i_{n_i}}\}^2 \setminus \{(v, v) \mid v \in X_i\} \to \mathcal{P}(\{1, \ldots, 4\}).$$

Nun können wir die Lösungsmenge $L^{z_{X_i}}$ aufstellen

$$\begin{aligned} L^{z_{X_i}} = \{&(z_{X_j}, z_{X_k}) \in Z^{z_{X_i}} \mid \forall u_1, v_1, u_2, v_2 \in X_i : z_{X_j}(u_1, v_1) \triangle_P z_{X_k}(v_1, u_1) \neq \dagger, \\ &(((z_{X_j}(u_1, v_1) \triangle_P p(v_1, u_2)) \triangle_P z_{X_k}(u_2, v_2)) \triangle_P p(v_2, u_1)) \neq \dagger\}. \end{aligned}$$

Für alle Paare $(z_{X_j}, z_{X_k}) \in L^{z_{X_i}}$ gibt es k-PGA's D_j und D_k, die jeweils z_{X_j} und z_{X_k} respektieren, und es gilt, dass $D_i(V(G_i), F(D_j) \cup F(D_k))$ eine k-PGA ist, die z_{X_i} respektiert. Mit dieser Formulierung ist es uns nun möglich aus den Einträgen von T_{X_j} und T_{X_k} diejenigen für T_{X_i} zu berechnen.

$$T_{X_i}(z_{X_i}) = \min_{(z_{X_j}, z_{X_k}) \in L^{z_{X_i}}} (T_{X_j}(z_{X_j}) + T_{X_k}(z_{X_k}) - g_{X_i}(z_{X_j}, z_{X_k}))$$

mit $g_{X_i}(z_{X_j}, z_{X_k}) = |\{u \mid u \in X_i,\ z_{X_j}(u) = z_{X_k}(u) = IV\}|$. Hiermit wird verhindert, dass Knoten, die sowohl in D_j als auch in D_k in der Bezugsmenge sind, doppelt gezählt werden.

Lemma 5.14. *Um für einen Join Node X_i die Tabelleneinträge in T_{X_i} zu konstruieren, benötigt man $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^3 \cdot k^4)$ Schritte.*

Beweis. Für alle Beutelzustände z_{X_i} haben die Mengen $Z^{z_{X_i}}$ und $L^{z_{X_i}}$ maximal $(16^{k^2} \cdot 3^{k^2} \cdot 5^2)^2$ Einträge. Für jeden dieser Einträge in $Z^{z_{X_i}}$ müssen $O(k^2)$ Schritte und für jeden Eintrag in $L^{z_{X_i}}$ $O(k^4)$ Schritte unternommen werden. Insgesamt für alle Zustände z_{X_i} benötigen wir also $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^3 \cdot k^4)$ Schritte. Die Abbildung p ist wieder in $O(16^{k^2} \cdot 3^{k^2} \cdot 5^2 \cdot k!)$ Schritten konstruierbar. Da wir zur Konstruktion der Tabelleneinträge alle Elemente aus $L^{z_{X_i}}$ für jeden Zustand z_{X_i} begutachten müssen, folgt eine totale Zeitkomplexität von $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^3 \cdot k^4)$. □

5.2.5 Ermittlung der Lösung

Wenn wir nun für die Wurzel r von T die Einträge in T_{X_r} berechnet haben, so wollen wir nun damit die Lösung bestimmen. In T_{X_r} ist für jeden Beutelzustand z_{X_r} die Größe der Bezugsmenge einer kleinstmöglichen k-PGA festgehalten, die z_{X_r} respektiert. Da eine k-PGA eine Verallgemeinerung einer GA darstellt, müssen wir unter diesen k-PGA's diejenigen finden, die GA's sind. Der Unterschied zwischen k-PGA's und GA's ist der, dass für die Knotenmenge

V_η, hier also X_r, die Anforderungen einer GA abgeschwächt sind. Insbesondere wird für alle $v \in X_r$, die nicht in der Bezugsmenge sind, nur $indeg_{F_u}(v) \leq 1$ gefordert, anstatt $indeg_{F_u}(v) = 1$ wie es für eine GA typisch ist. Wenn nun für eine k-PGA für alle $v \in X_r$ gilt, dass entweder v in der Bezugsmenge ist oder $indeg_{F_u}(v) = 1$ der Fall ist, dann ist diese k-PGA eine GA. Dies wird klar, wenn wir uns die Definition 4.5 vor Augen führen und uns in Erinnerung rufen, dass eine k-PGA ebenfalls keinen alternierenden Kreis enthält. Da für Knoten $u \in Y_r$ bzw. $u \in V_\theta$ bereits gilt, dass u entweder in der Bezugsmenge ist oder dass $indeg_{F_u}(u) = 1$ gilt, müssen wir noch kurz überprüfen, ob jeder Knoten auch auf einem Observationspfad liegt, der in einem Knoten aus der Bezugsmenge seinen Ursprung hat. Dies ist aber auch unmittelbar einsichtig, da wir sonst einen alternierenden Kreis haben, der in einer k-PGA nicht vorkommen kann. Die Lösung für GAKB ist dann

$$\min_{z_{X_r}} \left\{ T_{X_r}(z_{X_r}) \;\middle|\; \begin{array}{l} z_{X_r} \text{ ist Beutelzustand für } X_r \\ \forall v \in X_r : (z_{X_r}(v) = IV \vee indeg_{F_u}(v) = 1) \end{array} \right\}.$$

Da ein Beutelzustand z_{X_i} auch einen Kantenzustand und einen Knotenzustand beinhaltet, ist es in $O(k^2)$ Schritten möglich festzustellen, ob für alle Knoten $v \in X_r$ außerhalb der Bezugsmenge $indeg_{F_u}(v) = 1$ bezüglich z_{X_r} gilt. Insgesamt können wir also in $O(16^{k^2} \cdot 3^{k^2} \cdot 5^k \cdot k^2)$ Schritten die Größe der kleinstmöglichen GA ermitteln. Um die Bezugsmenge hierfür zu ermitteln, müssen wir für diesen Eintrag rekonstruieren, aus welchen Teillösungen dieser während des Verfahrens entstanden ist. Dazu müssen wir in einem Backtracking-Verfahren höchstens die $O(n)$ Beutel betrachten.

5.2.6 Korrektheit und Laufzeit

Theorem 5.15. *Der vorgestellte Algorithmus löst GAKB auf Graphen mit beschränkter Baumweite korrekt.*

Beweis. Anhand unserer Ausführungen in den Paragraphen über Insert, Forget und Join Nodes ist klar, dass wir für jeden Knoten i aus den k-PGA's seiner Kinder k-PGA's für G_i berechnen. Wir müssen jetzt noch zeigen, dass in allen drei Fällen unter diesen neu konstruierten k-PGA's auch eine zu finden ist, die den gleichen Zustand akzeptiert und die gleiche Größe wie eine k-PGA für G_i hat, die in einer kleinstmöglichen Lösung für G vorkommt. Wir wollen hier einen Induktion über den maximalen Abstand n eines Knotens i in T zu einem Blatt führen. Gilt $n = 0$, so ist i ein Blatt. Dann ist klar, dass wir den Zustand z_{X_i}, der zu einer k-PGA aus der kleinstmöglichen Lösung für G_i gehört, durch vollständige Suche gefunden haben und es gilt dann $T_{X_i}(z_{X_i}) < \infty$. Gelte nun die Behauptung für alle Knoten mit maximalem Abstand n zu einem Blatt. Sei i nun ein Knoten mit maximalem Abstand $n+1$. Für die Kinder von i gilt also bereits die Behauptung. Sei D_i die k-PGA für G_i, die in der kleinstmöglichen GA für G zu finden ist. Sei z_{X_i} der Zustand der von D_i respektiert wird. Wir unterscheiden für den Beutel X_i wieder drei Fälle:

Insert Nodes: Sei j das Kind von i. Wenn wir z_{X_i} auf G_j beschränken, ergibt sich ein Beutelzustand z_{X_j} der von D_j, der Beschränkung von D_i auf G_j, respektiert wird. Gilt $f_{z_{X_i}}(x) = 1$, dann gilt mit einem induktiven Argument $z_{X_j} \in L^{z_{X_i}}$ und $T_{X_j}(z_{X_j}) = q$, wobei q die Anzahl der Knoten

in der Bezugsmenge von D_i abzüglich 1 ist. Aus der Art der Konstruktion der Lösungen für Insert Nodes folgt dann $T_{X_i}(z_{X_i}) = T_{X_j}(z_{X_j}) = q + 1$. Gilt $f_{z_{X_i}} = 0$, so folgt induktiv $T_{X_j}(z_{X_j}) = q'$ und q' ist die Anzahl der Knoten in der Bezugsmenge von D_i. Dann folgt nach Konstruktion $T_{X_i}(z_{X_i}) = q'$.

Forget Nodes: Ist j das Kind von i, so sei D_j die k-PGA, wenn wir D_i auf G_j betrachten mit $V_\eta = X_j$ und $V_\theta = Y_j$. D_j respektiert einen Zustand z_{X_j} für den induktiv $T_{X_j}(z_{X_j}) = q$ und $z_{X_j} \in L^{z_{X_i}}$ gilt. Klar ist dann nach Konstruktion $T_{X_i}(z_{X_i}) = q$ und q ist die Größe der Bezugsmenge von D_i.

Join Node: Sind j und k die Kinder von i, so seien D_k und D_j wieder die k-PGA's, die wir erhalten, wenn wir D_i auf G_j bzw. G_k einschränken. Sei q die Anzahl der Knoten in der Bezugsmenge aus D_i. Seien z_{X_j} und z_{X_k} die entsprechenden Beutelzustände für D_j und D_k und q_j und q_k die Größen der entsprechenden Bezugsmengen. Dann muss $q = q_j + q_k - g$ gelten, wobei g die Knoten aus der Bezugsmenge von D_i sind, die in $D_i[X_i]$ liegen. Induktiv gilt $T_{X_j}(z_{X_j}) = q_j$, $T_{X_k}(z_{X_k}) = q_k$ und $(z_{X_j}, z_{X_k}) \in L^{z_{X_i}}$. Es folgt also nach Konstruktion $T_{X_i}(z_{X_i}) = q_j + q_k - g = q$, da wir die Knoten, die im Schnitt der Bezugsmenge von D_j und D_k liegen, abziehen. □

Theorem 5.16. *Der Algorithmus löst GAKB auf Graphen mit beschränkter Baumweite k in $O(16^{3k^2} \cdot 3^{3k^2} \cdot 5^{3k} \cdot k^4 \cdot n)$ Schritten.*

Beweis. Die dominierende Laufzeitkomplexität pro Knoten von T aus den Lemmata 5.11-5.14 ist $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2)^3 \cdot k^4)$. Da wir zur Ermittlung der Lösung aus den Tabelleneinträgen nur $O((16^{k^2} \cdot 3^{k^2} \cdot 5^2 \cdot k^2)k^2)$ Zeit brauchen und T insgesamt $O(n)$ Knoten besitzt, ist die Gesamtlaufzeit $O(16^{3k^2} \cdot 3^{3k^2} \cdot 5^{3k} \cdot k^4 \cdot n)$. □

Kapitel 6

Anmerkungen, Zusammenfassung und Ausblick

6.1 Verminderung des konstanten Laufzeitfaktors

In den beiden vorgestellten Algorithmen geht ein sehr hoher konstanter Faktor in die Laufzeit ein. Insbesondere beim Algorithmus für GSP-Graphen vermittelt die hohe Konstante den Eindruck, dass dieser nicht für die Praxis geeignet ist. Wir müssen für jeden inneren Knoten des Parse-Trees $25 \cdot 16 \cdot 16 = 6400$ 2-PGA's berechnen, abhängig davon, welche Terminalzustände herrschen und welche alternierenden Pfade zwischen den Terminalen existieren sollen. Dazu müssen wir in einer Implementierung ebenso alle 6400 Zustände der beiden Kinder jeweils miteinander komponieren, welches zu $6400^2 = 40960000$ Operationen führt. Aus diesem Grund ist dieser Algorithmus erst ab 26 Knoten besser als der naive Algorithmus.
Es besteht die Möglichkeit diese Konstante zu senken, da wir für zwei Terminalknoten die PGA's nicht bezüglich aller möglichen Kombinationen von Pfadklassen zwischen ihnen unterscheiden müssen. Beispielsweise ist es unerheblich, ob zwischen zwei Knoten ein Pfad der Klasse 1 verläuft, oder jeweils ein Pfad der Klasse 1 und 2. Denn jedesmal wenn ein Pfad der Klasse 2 mit einem anderen alternierenden Pfad einen alternierenden Kreis bildet, so tut dies dieser Pfad auch mit dem Pfad der Klasse 1. Aus diesem Grund legen wir Folgendes fest:

Definition 6.1. *Seien i, j Pfadklassen. Der Pfad i überlagert j, $i \prec j$ abgekürzt, wenn für alle $k \in \{1, \ldots, 4\}$ $k \oplus_P j = \dagger$ (bzw. $j \oplus_P k = \dagger$) $\Rightarrow k \oplus_P i = \dagger$ (bzw. $i \oplus_P k = \dagger$) gilt.*

Anhand Tabelle 4.4.2 erkennen wir, dass gilt:

$$1 \prec 1, 1 \prec 2, 1 \prec 3, 1 \prec 4, 2 \prec 2, 2 \prec 4, 3 \prec 3, 3 \prec 4, 4 \prec 4.$$

Sobald also ein alternierender Pfad P_1 einem anderen P_2 überlagert, so fügt P_2 keine weiteren Restriktionen hinzu, falls P_1 vorliegt und die gleiche Richtung

besitzt. Eine 2-PGA, die zwei alternierende Pfade besitzt, wobei der erste den zweiten überlagert, ist aufgrund des zweiten nicht von einer 2-PGA zu unterscheiden, die nur den ersten alternierenden Pfad hat. Deswegen wollen wir die Teilmengen von $\{1,\ldots,4\}$ in Klassen einteilen.

Definition 6.2. *Sei* $I \subseteq \{1,\ldots,4\}$. *Dann ist*

$$\begin{aligned} I_\prec \quad := \quad & \{\{j_{k_1},\ldots,j_{k_s}\} \mid I \subseteq \{j_{k_1},\ldots,j_{k_s}\} \subseteq \{1,\ldots,4\}, \\ & \forall h \in \{j_{k_1},\ldots,j_{k_s}\} : \exists i \in I : i \prec h\}. \end{aligned}$$

$I_\prec$ *heißt* Überlagerung.

Für alle $K, J \in I_\prec$ gilt: Genau dann gibt es ein $z \in \{1,\ldots,4\}$ und ein $k \in K$ mit $z \oplus_P k = \dagger$, wenn es es ein $j \in J$ mit $z \oplus_P j = \dagger$ gibt. Nehmen wir an, wir haben zwei 2-PGA's D_1, D_2 für den gleichen GSP-Subgraphen G' eines GSP-Graphen G. D_1 und D_2 unterscheiden sich so, dass in D_1 die Pfadklassen in K zwischen dem Terminal u und dem Terminal v verlaufen und in D_2 verlaufen die Pfadklassen aus J zwischen diesen Knoten. D_1 und D_2 müssen dann hinsichtlich der Restriktionen über ihre Weiterverwendung bezüglich einer Lösung für G nicht mehr unterscheiden werden. Haben D_1 und D_2 die gleichen Terminalzustände und enthalten beide die gleiche Anzahl an Knoten aus der Bezugsmenge, so können sie in einer etwaigen optimalen Lösungen gegeneinander ausgetauscht werden. Mit Definition 6.2 ergeben sich dann die folgenden inklusionsmaximalen und disjunkten Überlagerungen:

$$\begin{aligned} \{1\}_\prec &= \{\{1\},\{1,2\},\{1,3\},\{1,4\},\{1,2,3\},\{1,2,4\},\{1,3,4\},\{1,2,3,4\}\} \\ \{2\}_\prec &= \{\{2\}\{2,4\}\} \\ \{3\}_\prec &= \{\{3\},\{3,4\}\} \\ \{4\}_\prec &= \{\{4\}\} \\ \{2,3\}_\prec &= \{\{2,3\},\{2,3,4\}\} \\ \{\emptyset\}_\prec &= \{\emptyset\} \end{aligned}$$

Wir wollen die Klassifizierung der Pfadklassen zwischen zwei Knoten jetzt nur noch mittels der Überlagerungen $\{1\}_\prec, \{2\}_\prec, \{3\}_\prec, \{4\}_\prec, \{2,3\}_\prec$ und $\{\emptyset\}_\prec$ vornehmen. Dazu ist es nötig, dass auf ihnen die Operationen $\oplus_{S1}$ und $\oplus_P$ durchführbar sein müssen. Für $\oplus_P$ ist dies einfach. Falls K, J Überlagerungen sind, betrachten wir als erstes den Fall, dass für alle $j \in J, k \in K$ gilt, dass j und k verschiedene Richtungen haben. Für zwei Überlagerungen $K_\prec$ und $J_\prec$ müssen wir nur feststellen, ob es $k \in K$ und $j \in J$ mit $k \oplus_P j = \dagger$ gibt. Dies führt zu Tabelle 6.1.

Wenn aber für alle $j \in J, k \in K$ gilt, dass j und k die gleiche Richtung haben, ergibt sich ein zweiter Fall. Wir müssen untersuchen, welche Überlagerung entsteht, wenn wir J und K vereinigen. Dies muss jeweils die gleiche Überlagerung sein, unabhängig davon, welche Pfadmengen sich hinter J und K verbergen. Tabelle 6.2 gibt für alle möglichen Überlagerungen, welche J und Kannehmenn können, Aufschluss hierüber.

Etwas aufwendiger gestaltet es sich für die Operation $\oplus_{S1}$. Wenn wir hier $K_\prec \oplus_{S1} J_\prec = I_\prec$ für Überlagerungen K, J und I schreiben wollen, so muss dies auch für die einzelnen Bestandteile von K, J und I gelten. Es gilt also $K_\prec \oplus_{S1} J_\prec = I_\prec$ genau dann, wenn es für alle $k \in K$ und $j \in J$ ein $i \in I$

$\oplus_P$	$\{1\}_\prec$	$\{2\}_\prec$	$\{3\}_\prec$	$\{4\}_\prec$	$\{2,3\}_\prec$	$\{\emptyset\}_\prec$
$\{1\}_\prec$	†	†	†	†	†	$*$
$\{2\}_\prec$	†	†	$*$	$*$	†	$*$
$\{3\}_\prec$	†	$*$	†	$*$	†	$*$
$\{4\}_\prec$	†	$*$	$*$	$*$	$*$	$*$
$\{2,3\}_\prec$	†	†	†	$*$	†	$*$
$\{\emptyset\}_\prec$	$*$	$*$	$*$	$*$	$*$	$*$

Tabelle 6.1: *Gibt an, welche Überlagerungen durch eine P-Komposition alternierende Kreise bilden. Ein Eintrag † bedeutet, dass einer Zustande kommt, ein $*$, dass keiner entsteht. Hierbei haben die Pfadmengen, für die die Überlagerungen stehen, verschiedene Richtungen.*

$\oplus_P$	$\{1\}_\prec$	$\{2\}_\prec$	$\{3\}_\prec$	$\{4\}_\prec$	$\{2,3\}_\prec$	$\{\emptyset\}_\prec$
$\{1\}_\prec$	$\{1\}_\prec$	$\{1\}_\prec$	$\{1\}_\prec$	$\{1\}_\prec$	$\{1\}_\prec$	$\{1\}_\prec$
$\{2\}_\prec$	$\{1\}_\prec$	$\{2\}_\prec$	$\{2,3\}_\prec$	$\{2\}_\prec$	$\{2,3\}_\prec$	$\{2\}_\prec$
$\{3\}_\prec$	$\{1\}_\prec$	$\{2,3\}_\prec$	$\{3\}_\prec$	$\{3\}_\prec$	$\{2,3\}_\prec$	$\{3\}_\prec$
$\{4\}_\prec$	$\{1\}_\prec$	$\{2\}_\prec$	$\{3\}_\prec$	$\{4\}_\prec$	$\{2,3\}_\prec$	$\{4\}_\prec$
$\{2,3\}_\prec$	$\{1\}_\prec$	$\{2,3\}_\prec$	$\{2,3\}_\prec$	$\{2,3\}_\prec$	$\{2,3\}_\prec$	$\{2,3\}_\prec$
$\{\emptyset\}_\prec$	$\{1\}_\prec$	$\{2\}_\prec$	$\{3\}_\prec$	$\{4\}_\prec$	$\{2,3\}_\prec$	$\{\emptyset\}_\prec$

Tabelle 6.2: *Gibt an, welche Überlagerungen durch eine P-Komposition entstehen. Die jeweiligen Pfadmengen, für die die Überlagerungen stehen, haben die gleiche Richtung.*

gibt mit $k \oplus_{S1} j = i$. Wenn wir dies für alle Klassen $\{1\}_\prec, \{2\}_\prec, \{3\}_\prec, \{4\}_\prec$, $\{2,3\}_\prec$ und $\{\emptyset\}_\prec$ untersuchen, so ergibt sich für $\oplus_{S1}$ Tabelle 6.3. Ebenfalls sind die alternierenden Pfade, die im Basisgraph, der einzelnen Kante also, für GSP-Graphen vorkommen, in Überlagerungen einteilbar. Dann müssen wir die Knotenpaarzustände anders definieren. Nämlich so, dass ein Knotenpaar (u, v) auf die Überlagerungen abgebildet wird. Also muss für einen GSP-Graphen $G(V, E, u, v)$ dann

$$z_{X_i:p} : \{(u,v),(v,u)\} \to \{\{1\}_\prec, \{2\}_\prec, \{3\}_\prec, \{4\}_\prec, \{2,3\}_\prec, \{\emptyset\}_\prec\}$$

festgelegt werden. Dementsprechend muss in einem Tabelleneintrag nicht mehr unterschieden werden, welche alternierenden Pfade von einem zum anderen Terminal verlaufen, sondern in welche Überlagerung diese Menge von alternierenden Pfaden gehört. Die Tabelle für einen GSP-Graphen $G(V, E, u, v)$ sieht dann folgendermaßen aus:

$$T_G(z) = \min_{A \subseteq V} \left\{ |A| \;\middle|\; \begin{array}{l} \exists \text{ 2-PGA } D(V, F, a, b) \text{ bzgl. } A, z_D(u) = a, z_D(v) = b \\ \exists I_1 \in z_D(u,v), I_2 \in z_D(v,u)\ \forall s_1 \in I_1, s_2 \in I_2 \\ \exists \text{ alt. Pfad vom Typ } s_1 \text{ bzw. } s_2 \text{ zw. } u, v \text{ bzw. zw. } v, u \end{array} \right\}$$

Da wir die Operationen $\oplus_P$ und $\oplus_{S1}$ definieren konnten, so können wir die 'Pfadarithmetik' nun durch 'Überlagerungsarithmetik' im Algorithmus ersetzen. Damit verringert sich die Konstante erheblich, denn wir unterscheiden eine 2-PGA nur noch durch Terminalzustände und Überlagerungen bezüglich der

$\oplus_{S1}$	$\{1\}_\prec$	$\{2\}_\prec$	$\{3\}_\prec$	$\{4\}_\prec$	$\{2,3\}_\prec$	$\{\emptyset\}_\prec$
$\{1\}_\prec$	$\{1\}_\prec$	$\{1\}_\prec$	$\{3\}_\prec$	$\{3\}_\prec$	$\{1\}_\prec$	$\{\emptyset\}_\prec$
$\{2\}_\prec$	$\{2\}_\prec$	$\{2\}_\prec$	$\{4\}_\prec$	$\{4\}_\prec$	$\{2\}_\prec$	$\{\emptyset\}_\prec$
$\{3\}_\prec$	$\{1\}_\prec$	$\{\emptyset\}_\prec$	$\{3\}_\prec$	$\{\emptyset\}_\prec$	$\{3\}_\prec$	$\{\emptyset\}_\prec$
$\{4\}_\prec$	$\{2\}_\prec$	$\{\emptyset\}_\prec$	$\{4\}_\prec$	$\{\emptyset\}_\prec$	$\{4\}_\prec$	$\{\emptyset\}_\prec$
$\{2,3\}_\prec$	$\{1\}_\prec$	$\{2\}_\prec$	$\{3\}_\prec$	$\{4\}_\prec$	$\{2,3\}_\prec$	$\{\emptyset\}_\prec$
$\{\emptyset\}_\prec$	$\{\emptyset\}_\prec$	$\{\emptyset\}_\prec$	$\{\emptyset\}_\prec$	$\{\emptyset\}_\prec$	$\{\emptyset\}_\prec$	$\{\emptyset\}_\prec$

Tabelle 6.3: *Gibt an welche Überlagerung durch die S1-Komposition zweier weiterer entsteht.*

Terminale. Damit müssen wir pro innerem Knoten des Parse-Trees nur noch $25 \cdot 6 \cdot 6 = 900$ Einträge berechnen. Diese Einträge berechnet man nun dadurch, dass wir wieder alle Lösungen der beiden Kinder kombinieren müssen. Dies erfordert $900^2 = 810000$ Operationen. Mit analogen Modifikationen ist es dann auch möglich den Algorithmus für Graphen mit beschränkter Baumweite zu verbessern. Wir können so den konstanten, von k abhängigen Faktor senken und PDS dann in $O(6^{3k^2} \cdot 3^{3k^2} \cdot 5^{3k} \cdot k^2 \cdot n)$ lösen.

6.2 Zusammenfassung und Ausblick

In dieser Arbeit wurden im Wesentlichen zwei neue Ergebnisse bezüglich Power Domination vorgestellt. Das erste Ergebnis ist die Konstruktion einer Reduktion von Dominating Set auf Power Dominating Set. Ausgehend von dieser Reduktion haben wir für verschiedene Graphklassen nachweisen können, dass Power Dominating Set **NP**-vollständig bleibt, wenn wir es auf bestimmte Klassen beschränken. Unter diesen Graphklassen waren die planaren Graphen, Split-Graphen und Circle-Graphen. Hiermit ist für alle Graphklassen, für die in [17] Dominating Set **NP**-vollständig ist, ebenfalls Power Dominating Set **NP**-vollständig. Weitere Folgerungen der Reduktion für Power Dominating Set waren die $W[2]$-Härte und eine untere Schranke von $\Theta(\log n)$ für die Approximationsgüte.
Das zweite Ergebnis umfasst die Neuformulierung von Power Dominating Set als Gültige Ausrichtung mit kleinstmöglicher Bezugsmenge und die Entwicklung von Linearzeit-Algorithmen für GSP-Graphen und für Graphen mit beschränkter Baumweite. In [17] wird die Klasse der Partial k-Trees als in polynomieller Zeit lösbar für Dominating Set angegeben. Diese Klasse umfasst genau die Graphen mit beschränkter Baumweite k. Damit haben wir die erste Graphklasse aus [17] gefunden, für die sowohl Dominating Set als auch Power Dominating Set für festes k in **P** liegen. Die Formulierung von Power Dominating Set als GAKB scheint aus algorithmischer Sicht eine bessere Charakterisierung des Problems zu sein und lässt eher erahnen, wo die inhärente Schwierigkeit dieses Problems liegt.
Erwähnt soll auch der vereinfachte lineare Algorithmus für Bäume werden, der deutlich weniger komplex ist als derjenige in [10].
Ausgehend von diesen Ergebnissen ist es möglich, viele verschiedene Richtungen einzuschlagen, die vielversprechend für zukünftige Forschung sind. In Tabelle 2.2 bleiben einige Einträge leer, die die Komplexität von Power Dominating Set

betreffen und die wir nicht behandelt haben. Allerdings wurden in [17] für die entsprechenden Graphklassen angegeben, dass Dominating Set in **P** liegt. Hier stellt sich die Frage, ob es einen signifikanten Unterschied zwischen der Komplexität von Dominating Set und Power Dominating Set gibt. Im Speziellen bleibt offen, ob es eine Graphklasse gibt, für die Dominating Set in **P** liegt und Power Dominating Set **NP**-vollständig ist oder ob es sich gerade umgekehrt verhält. Weiter ist ebenso von Interesse, ob es eine Graphklasse gibt, die eine bessere Approximationgüte für Power Dominating Set erlaubt, als eine mit Faktor $\Theta(\log n)$. Für Dominating Set ist bekannt, dass es auf planaren Graphen fixed-parameter tractable bezüglich der Lösungsgröße ist. Lässt sich dieses Ergebnis ebenfalls auf Power Dominating Set übertragen? In [1] werden nicht-triviale Datenreduktionsregeln für DS angegeben. Sind diese Regeln nicht mehr anwendbar auf den gegebenen Graphen, so folgt für planare Graphen, dass dieser Problemkern eine Größe hat, die linear abhängig von der Größe der kleinstmöglichen Dominating Set ist. Gibt es solche Datenreduktionsregeln für Power Dominating Set, die die gleichen Auswirkungen auf planare Graphen haben? Die Umformulierung von Power Dominating Set als GAKB verspricht hilfreich zu sein, um Antworten auf diese Fragen zu finden.

Literaturverzeichnis

[1] Jochen Alber, Michael R. Fellows, and Rolf Niedermeier. Polynomial-time data reduction for dominating set. *Journal of the ACM*, 51(3):363–384, 2004. 18, 94

[2] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal Algebraic Discrete Methods*, 8:277–284, 1987. 74

[3] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation—Combinatorial Optimization Problems and their Approximability Properties*. Springer-Verlag, 1999. 18, 25, 26

[4] H.L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996. 74

[5] H.L. Bodlaender. Algorithmic techniques and results. In *In Proceedings 22nd MFCS*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997. 73

[6] H.L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. Technical Report UU-CS-1996-2, Department of Theoretical Computer Science, Utrecht Univerity, 1998. 41

[7] R.G. Downey and M.R. Fellows. *Parametrized Complexity*. Springer, 1999. 18

[8] R.J. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10:303–318, 1965. 41

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completness*. W. H. Freeman, San Francisco, 1979. 21, 22

[10] Theresa W. Haynes, Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Michael A. Henning. Domination in graphs applied to electric power networks. *SIAM Journal on Discrete Mathematics*, 15(4):519–529, 2003. 1, 17, 19, 28, 93

[11] T.W. Haynes, S.T. Hedetniemi, and P.J. Slater. *Fundamentals of Domination in Graphs*. Marcel Dekker, New York, 1998. 17

[12] J.M. Keil. The complexity of domination problems in circle graphs. *Discrete Applied Mathematics*, 42:51–63, 1993. 23

[13] T. Kikuno, N. Yoshida, and Y. Kakuda. A linear algorithm for the domination number of a series-parallel graph. *Discrete Applied Mathematics*, 5:299–311, 1983. 43, 44, 55, 71

[14] T. Kloks. *Treewidth: Computations and Approximations*, volume 842 of *LNCS*. Springer, 1994. 73, 74

[15] J. Kneis, D. Moelle, P. Richter, and P. Rossmanith. Parameterized power domination complexity. Technical Report AIB-2004-9, Fakultät für Informatik, RWTH Aachen, 2004. 15, 19

[16] N.M. Korneyenko. Combinatorial algorithms on a class of graphs. *Discrete Applied Mathematics*, 54:215–217, 1994. 43

[17] D. Kratsch. Algorithms. In *Domination in Graphs: Advanced Topics*, pages 191–231. Marcel Dekker, New York, 1998. 14, 19, 26, 93, 94

[18] M.A. Henning M. Dorfling. A note on power domination in grid graphs, 2004. manuskript. 17

[19] N. Robertson and P.D. Seymour. Graph minors II. Algortihmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986. 73

[20] K. Takamizawa, T. Nishizeki, and N. Saito. Linear computability of combinatorial problems on series-parallel graphs. *Journal of the ACM*, 29(3):623–641, 1982. 41, 43, 71

[21] J.A. Wald and C.J. Colburn. Steiner trees, partial 2-trees and minimum IFI networks. *Networks*, 14:159–167, 1983. 41