Boolsche Weite:

Analyse, Schranken & Lösbarkeit

Masterarbeit

im Masterstudiengang Mathematik

vorgelegt von:	Björn Böken
Matrikelnummer:	280047
Erstgutachter:	Prof. Dr. Arie M.C.A. Koster
Zweitgutachter:	Priv.-Doz. Dr. Walter Unger

Zusammenfassung

Baumweite ist ein nützliches Werkzeug, um schwierige Probleme auf Graphen effizient lösen zu können. Da das Bestimmen der Baumweite eines gegebenen Graphen G allgemein $\mathcal{NP}$-schwer ist, sind daher schnelle und gute Approximationen wünschenswert. Entsprechend überrascht es nicht, dass sich in den letzten Jahren viele Arbeiten den diversen Seiten der Baumweite gewidmet haben. Ziel ist es nun, eine mit der Baumweite verwandte Größe vorzustellen und zu untersuchen - die Boolsche Weite.

Abstract

Boolsche Weite ist eine Größe, die ähnliche Eigenschaften besitzt wie die Baumweite, allerdings höchstens so groß ist wie sie. Tatsächlich existieren Graphen, deren Boolsche Weite nur logarithmisch in ihrer Baumweite und Cliquenweite ist. Dies ergibt eine essentielle Verbesserung bei entsprechenden Algorithmen, von exponentiell in der Baumweite zu linear bzw. polynomial in der Boolschen Weite. Es wird daher versucht, einige bekannte Schranken an die Baumweite auf Boolsche Weite zu übertragen und u.a. wird gezeigt, dass die Boolsche Weite eines induzierten Teilgraphen von G höchstens so groß wie $boolw(G)$ selbst ist, während gleichzeitig nachgewiesen wird, dass dies für allgemeine Graphen nicht gilt (da $boolw(K_n) = 1$). Weiter wird gezeigt, dass sich die Boolsche Weite von $(k \times k)$-Gittergraphen G_k für $k \to \infty$ einem Wert $\approx 0.81137 \cdot k - 0.46968$ beliebig nähert. Darüber hinaus werden mit den H-Gittern und I-Gittern zwei weitere Klassen perfekter Graphen auf k^2 Knoten vorgestellt und es wird gezeigt, dass beide Graphen Boolsche Weite $\Theta(\log(k))$ haben, während ihre Baumweite in $\Omega(k)$ und ihre Cliquenweite in $\Theta(k)$ liegt. Schließlich wird ein ganzzahliges lineares Programm vorgestellt, mit dessen Hilfe sich die Boolsche Weite für allgemeine Graphen bestimmen lässt, das aber leider nur für kleine Graphen praktikabel ist.

Inhaltsverzeichnis

Abkürzungsverzeichnis

$\alpha(G)$	Stabile-Mengenzahl von G
$\Delta(G)$	Maximalgrad von G
$\delta(G)$	Minimalgrad von G
$\mathcal{NP}$	Klasse der Probleme, die nichtdeterministisch-polynomiell lösbar sind
$\mathcal{P}$	Klasse der Probleme, die deterministisch-polynomiell lösbar sind
$\nu(G)$	Matchingzahl von G
$\omega(G)$	Cliquenzahl von G
$\overline{M}$, $\overline{G}$	Komplement der Menge M bzw. Komplementgraph von G
$boolw(G)$	Boolsche Weite von G (vgl. 3.3.4)
$cw(G)$	Cliquenweite von G (vgl. 3.2.6)
$d_G(v)$	Grad von Knoten v in G
$diam(G)$	Durchmesser von G (maximale Länge eines Pfades)
$dist_G(v, w)$	Distanz von v und w in G
$E(G)$	Kantenmenge von G
$g(G)$	Taillenweite von G (minimale Länge eines Kreises)
$N(a)$, $N(A)$	offene Nachbarschaft von Knoten a bzw. Knotenmenge A
$N[a]$, $N[A]$	abgeschlossene Nachbarschaft von Knoten a bzw. Knotenmenge A
$tw(G)$	Baumweite von G (vgl. 3.1.1)
$V(G)$	Knotenmenge von G

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung & Motivation

Viele $\mathcal{NP}$-vollständige und $\mathcal{NP}$-schwere Probleme auf Graphen effizient zu lösen ist eines der zentralen Forschungsgebiete der modernen Mathematik sowie Informatik und mündet nicht zuletzt in der Frage, ob dies grundsätzlich möglich ist oder nicht, sprich ob $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{P} \neq \mathcal{NP}$ gilt. Leider ist diese Frage immer noch offen, wenngleich $\mathcal{P} \neq \mathcal{NP}$ die verbreitete Annahme ist.

Entsprechend ist nach aktuellem Stand davon auszugehen, dass die meisten dieser Probleme auf Graphen "schwer" sind und Lösungsmethoden daher einen guten Ansatz brauchen, wenngleich sich exponentielle Laufzeiten unter $\mathcal{P} \neq \mathcal{NP}$ für den allgemeinen Fall nicht verhindern lassen.

Ein wichtiger Schritt in dieser Richtung ist der Begriff der Baumweite eines Graphen. Da insbesondere auf Bäumen viele schwere Probleme sehr einfach werden können, ist die Baumweite ein gutes Maß dafür, wie "ähnlich" die Struktur eines Graphen zu der eines Baumes ist. Tatsächlich ist dies auch für viele in der Praxis relevante Probleme von Interesse: Denn lässt sich diese durch eine Konstante k beschränken bzw. eine zulässige Baumzerlegung mit Weite k für einen gegebenen Graphen angeben, werden viele Probleme in Linearzeit lösbar, siehe Arnborg und Proskurowski [1989] oder Bodlaender [1988].

Entsprechende Algorithmen haben - unabhängig vom jeweiligen Problem - gemeinsam, dass zunächst eine Baumzerlegung mit Baumweite k bestimmt wird und dann, mit dynamischer Programmierung oder über Divide-&-Conquer-Algorithmen, das eigentliche Problem effizient über die Baumzerlegung gelöst wird.

Die Komplexität dieser Algorithmen ist dann üblicherweise zwar linear bzw. polynomial, dennoch normalerweise exponentiell in der Baumweite k, entsprechend sind zulässige Baumzerlegungen mit möglichst kleinem - bestenfalls optimalem - k wünschenswert. Andererseits ist das Bestimmen einer Baumzerlegung mit minimaler Weite ebenfalls $\mathcal{NP}$-schwer, vgl. Arnborg und Proskurowski [1986]. Daher sind effizient zu berechnende, gute Approximationen an die Baumweite erwünscht bzw. notwendig, wenn $\mathcal{NP}$-vollständige oder $\mathcal{NP}$-schwere Probleme derart gelöst werden sollen.

Ein großer Nachteil der Baumweite ist allerdings, dass sie durch große vollständige Teilgraphen unweigerlich wächst, da $tw(G) \geq \omega(G) - 1$ gilt. Eine Verbesserung in diese Richtung ist die sogenannte Cliquenweite, vgl. Golumbic und Rotics [2000] oder Lozin [2008]. Hierbei wird versucht einen Graphen aus elementaren Operationen heraus zu konstruieren und es ergeben sich ähnliche Vorteile wie bei der Baumweite. Allerdings gibt es Graphen, deren Cliquenweite exponentiell in ihrer Baumweite ist (siehe Corneil und Rotics [2001]), während gleichzeitig gezeigt werden kann, dass diese Schranke scharf ist bis auf Multiplikation mit Konstanten, also die Cliquenweite eines Graphen immer höchstens exponentiell in seiner Baumweite ist, vgl. Courcelle und Olariu [1997].

Boolsche Weite ist nun eine untere Schranke, sowohl an die Baumweite als auch an die Cliquenweite (siehe Bui-Xuan u. a. [2009]), die ebenfalls ähnliche Vorteile besitzt, bisweilen allerdings wesentlich kleiner ist. Für solche Graphen verbessert sich entsprechend die Performance von Algorithmen, die exponentiell in der Baumweite bzw. Boolschen Weite sind, massiv zugunsten der Boolschen Weite, je stärker die jeweiligen Größen wachsen.

Konkrete Beispiele für diese Situationen sind wie folgt gegeben: Einerseits ist es mit $n = |V(G)|$ möglich, das maximale gewichtete stabile Mengenproblem in $\mathcal{O}(n \cdot 2^{tw(G)})$ und das minimale gewichtete Dominanzmengenproblem in $\mathcal{O}(n \cdot tw(G)^2 \cdot 3^{tw(G)})$ zu lösen, vgl. Hvidevold u. a. [2011] und van Rooij u. a. [2009]. Andererseits kann nach Bui-Xuan u. a. [2009] das maximale gewichtete stabile Mengenproblem ebenso in $\mathcal{O}(n^2 \cdot boolw(G) \cdot 2^{2 \cdot boolw(G)})$ und das minimale gewichtete Dominanzmengenproblem in $\mathcal{O}(n^2 + n \cdot boolw(G) \cdot 2^{3 \cdot boolw(G)})$ gelöst werden. Weitere Beispiele sind in Adler u. a. [2010] zu finden.

Ist nun G derart, dass $boolw(G) \in \mathcal{O}(\log(tw(G))$ gilt, so wird die Komplexität der entsprechenden Algorithmen signifikant besser: Von exponentiell in der Baumweite $tw(G)$ zu linear bzw. polynomial in der Boolschen Weite $boolw(G)$. Hiervon ist eine große Menge von Graphklassen betroffen, u.a. Intervallgraphen, Permutationsgraphen, konvexe Graphen und Komplemente planarer Graphen, vgl. Belmonte und Vatshelle [2011]. Ferner gilt $cw(G) \in \Omega(\sqrt{n})$ für viele dieser Graphen.

Ähnlich ist die Situation auch bei Zufallsgraphen $G(n, p)$, deren Boolsche Weite nach Adler u. a. [2010] fast sicher in $\Theta(\log^2(n))$ liegt, während andererseits gleichzeitig $tw(G(n,p)) \in \Theta(n)$ und $cw(G(n,p)) \in \Theta(n)$ fast sicher gilt, vgl. Kloks und Bodlaender [1992] sowie Johansson [1998]. Weitere interessante Beispiele für derartige Graphen werden in Kapitel 4 folgen.

Alle diese Beispiele zeigen, dass Boolsche Weite bei vielen geeigneten Graphen wesentlich bessere Ergebnisse liefert als es die Baumweite oder Cliquenweite tun und es daher sinnvoll ist, Schranken an die Boolsche Weite zu untersuchen. Andererseits ist auch die Frage interessant, ob und mit welchem Aufwand, sich die Boolsche Weite eines Graphen explizit bestimmen lässt. Diesen Fragen soll in dieser Arbeit nachgegangen werden.

Die Arbeit ist dabei wie folgt gegliedert: Kapitel 2 stellt die verwendete Notation vor und führt die benötigten Grundlagen ein. Anschließend werden im 3. Kapitel die Begriffe Baumweite, Cliquenweite und Boolsche Weite definiert, einige grundlegenden Aussagen vorgestellt und bekannte Schranken an Baumweite auf Gültigkeit für Boolsche Weite hin überprüft. Im 4. Kapitel werden einige Graphklassen mit bekannter Baumweite und Cliquenweite auf ihre Boolsche Weite hin analysiert und abschließend wird in Kapitel 5 untersucht, in wie Weit sich die Boolsche Weite allgemeiner Graphen mit ganzzahliger linearer Programmierung bestimmen lässt.

2 Grundlagen & Notation

Die Arbeit beschäftigt sich mit der Analyse von endlichen, schlichten und ungerichteten Graphen und verwendet weitestgehend übliche Notationen. Diese sollen hier dennoch kurz vorgestellt werden. Eine ausführlichere Einführung ist in Standardwerken zur Graphentheorie, wie etwa Volkmann [2006] oder Diestel [2000], zu finden.

2.1 Knoten, Kanten, Grad und Nachbarschaften

Ist $G = (V, E)$ ein Graph, so bezeichnet $V = V(G) = \{v_1, v_2, \ldots, v_n\}$ die Knotenmenge und $E = E(G)$ die Kantenmenge. Da hier G immer schlicht ist, enthält E dabei weder Schlingen (also Kanten der Form $\{v, v\}$) noch Mehrfachkanten, also $E \subseteq \{\{v_i, v_j\} : 1 \leq i < j \leq n\}$.

Die Knoten $v, w \in V(G)$ heißen genau dann **benachbart** oder **adjazent**, falls $\{v, w\} \in E(G)$ ist. In dem Fall ist die Kante $\{v, w\}$ sowohl mit v als auch mit w **inzident**.

Für einen Knoten v ist $N(v)$ seine (offene) **Nachbarschaft**

$$N(v) := \{w \in V(G) : \{v, w\} \in E(G)\},$$

d.h. die Menge seiner Nachbarn. Analog ist seine **abgeschlossene Nachbarschaft** definiert als:

$$N[v] := \{w \in V(G) : \{v, w\} \in E(G)\} \cup \{v\}$$

Beide Definitionen lassen sich leicht auf folgende Art auf Knotenmengen $U \subseteq V(G)$ verallgemeinern:

$$N(U) := \left(\bigcup_{v \in U} N(v) \right) \setminus U$$

$$N[U] := \left(\bigcup_{v \in U} N(v) \right) \cup U$$

Weiter lässt sich mit Hilfe der Nachbarschaft auch sehr einfach der **Grad** eines Knotens definieren: Ist $v \in V(G)$, so ist der Grad eines Knotens $v \in V(G)$ definiert als die Anzahl seiner Nachbarn: $d(v) := |N(v)|$. Damit lässt sich für jeden Graphen G durch $\delta(G) := \min_{v \in V(G)} d_G(v)$ der **Minimalgrad** und $\Delta(G) := \max_{v \in V(G)} d_G(v)$ der **Maximalgrad** von G definieren.

Sind $v, w \in V(G)$ Knoten, so heißt die Länge, d.h. die Anzahl der Kanten, eines kürzesten v-w-Weges die **Distanz** von v und w und wird mit $dist(v, w)$ bezeichnet. Gibt es keinen v-w-Weg in G, so wird $dist(v, w) := \infty$ definiert. Der **Durchmesser** eines Graphen ist die maximale Distanz, die zwei Knoten zueinander haben und wird mit $diam(G)$ bezeichnet:

$$diam(G) := \max_{v,w \in V(G)} dist(v, w)$$

Enthält ein Graph G mindestens einen Kreis, so ist die **Taillenweite** von G die Länge eines kürzesten Kreises und wird mit $g(G)$ bezeichnet. Gibt es keinen Kreis in G, so wird $g(G) := \infty$ definiert. G heißt **zusammenhängend**, wenn er endlichen Durchmesser hat, also wenn je zwei Knoten zueinander endliche Distanz haben:

$$G \text{ zusammenhängend} :\Longleftrightarrow diam(G) < \infty \Longleftrightarrow \forall v, w \in V(G) : dist(v, w) < \infty$$

2.2 Schnitte, Teilgraphen, Komponenten und Minoren

Wieder sei $G = (V, E)$ ein Graph mit $A, B \subseteq V$. Das Paar (A, B) heißt ein **Schnitt** in G, falls $V = A \uplus B$. Es ist klar, dass dann $B = V \backslash A$ ist. Daher wird das **Komplement** von A definiert als $\overline{A} := V \backslash A$. Für jedes $A \subseteq V$ ist damit insbesondere $V = A \uplus \overline{A}$ und $(A, \overline{A})$ heißt der von A **induzierte Schnitt**.

Diese Definition lässt sich leicht auf Graphen verallgemeinern: Wenn $G = (V, E)$ ein Graph ist, so heißt der Graph $\overline{G} := (V, \overline{E})$ mit

$$\overline{E} := \{\, \{v, w\} \mid v, w \in V,\ v \neq w : \{v, w\} \notin E \,\}$$

der **Komplementärgraph** von G.

Ist $H = (W, F)$ ein weiterer Graph mit $W \subseteq V$ und $F \subseteq E$, so heißt H ein **Teilgraph** von G und es ist dann $H \leq G$. Für eine Knotenmenge $U \subseteq V$ ist $G[U] \leq G$ der von U **induzierte Teilgraph** mit:

$$V(G[U]) := U$$
$$E(G[U]) := E(G) \cap (U \times U)$$

$G[U]$ besteht also aus allen Knoten in U und allen Kanten aus G, die Knoten aus U miteinander verbinden. Ein Teilgraph $G' \leq G$ heißt dabei **induziert**, wenn er durch seine Knotenmenge induziert wird. Insbesondere ist jeder induzierte Teilgraph auch ein Teilgraph, aber es ist nicht jeder Teilgraph induziert.

Ferner heißt eine Knotenmenge $U \subseteq V$ **zusammenhängend** genau dann, wenn $G[U]$ zusammenhängend ist. Ist G nicht zusammenhängend, so heißen alle maximal-zusammenhängenden induzierten Teilgraphen **Komponenten** von G.

Ist weiter $v \in V(G)$ ein Knoten, so ist $G - v := G[V \backslash \{v\}]$ und analog für $U \subseteq V(G)$ ist $G - U := G[V \backslash U]$. Es seien nun $x, y \in V(G)$ Knoten mit $e = \{x, y\} \in E$. Dann entsteht H aus G durch **Kontraktion** von x und y, indem x und y aus G gelöscht werden und ein neuer Knoten u konstruiert wird, der zu allen Knoten benachbart ist, die vorher zu x oder zu y adjazent waren. Formal heißt das:

$$V(H) := V(G) \setminus \{x, y\} \uplus \{u\}$$
$$E(H) := E(G[V(H)]) \quad \uplus \{\ \{u, v\} \mid v \in N_G(\{x, y\})\ \}$$

Ein Graph M, der durch Löschen beliebig vieler Knoten und/oder Kanten sowie beliebig vieler Kantenkontraktionen aus G entsteht, heißt ein **Minor** von G. Diese Relation ist nach Definition offensichtlich transitiv, reflexiv und antisymmetrisch. Insbesondere ist damit jeder Teilgraph H von G auch ein Minor.

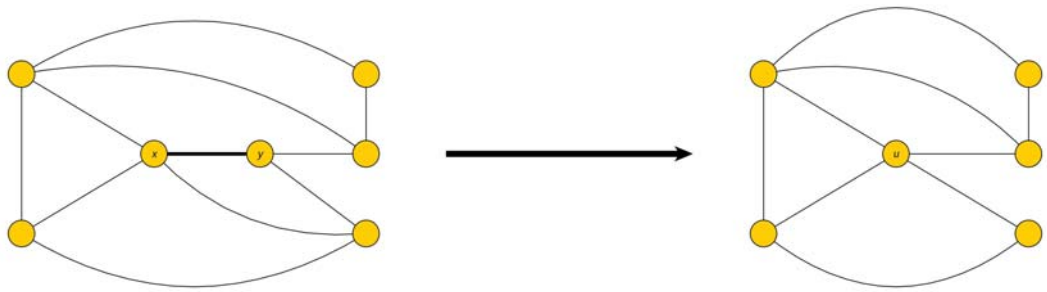

Abbildung 2.1: Kontraktion von x und y zu u

2.3 Bäume und Wälder

Besondere Graphen sind die, die keine Kreise enthalten. Ein solcher Graph heißt **Wald** und ein zusammenhängender Wald heißt **Baum**. Jeder Wald $T = (V, E)$ erfüllt, dass $0 \leq |E| \leq |V| - 1$ gilt, mit $|E| = |V| - 1$ genau dann, wenn T ein Baum ist. Die Knoten mit $d(v) = 1$ heißen **Blätter** und werden mit $\mathcal{L} = \mathcal{L}(T)$ bezeichnet:

$$\mathcal{L}(T) = \{v \in V(T) : d(v) = 1\}$$

Jeder Baum $T = (V, E)$ besitzt eine Vorgänger-/Nachfolgerrelation $\mathcal{R} \subset V \times V$ mit folgenden Eigenschaften:

- Ist $(v, w) \in \mathcal{R}$, so folgt $\{v, w\} \in E$. Insbesondere gilt $(v, v) \notin \mathcal{R}$ für alle $v \in V$.
- R ist **asymmetrisch**, d.h. $(v, w) \in \mathcal{R}$ impliziert $(w, v) \notin \mathcal{R}$.
- Es gibt genau einen Knoten $r \in V$ zu $\mathcal{R}$, so dass es für jedes $w \in V \backslash \{r\}$ ein eindeutiges $v \in V$ gibt mit $(v, w) \in \mathcal{R}$. Dieser Knoten r heißt **Wurzel** und der Knoten v ist dann **Vorgänger** oder **Vater** von w. Entsprechend heißt w **Nachfolger** oder **Sohn** von v.
- Ein Knoten ist genau dann ein Blatt, wenn er keine Nachfolger hat. Ist also $v \in V$ fest, so gilt $(v, w) \notin \mathcal{R}$ für alle $w \in V$ genau dann, wenn $v \in \mathcal{L}(T)$ ist.

Die Relation $\mathcal{R}$ ist durch diese Eigenschaften im Allgemeinen nicht eindeutig bestimmt und jedes $\mathcal{R}$ induziert dann einen gerichteten Baum $T = (V, \mathcal{R})$. Ist G ein Baum und existiert mit $\mathcal{R}$ eine solche Relation auf G, so dass für alle $v \in V(G)$ ferner $|\{w \in V(G) : (v, w) \in \mathcal{R}\}| \leq 2$ gilt, so heißt G ein **Binärbaum**.

Lässt sich die Bedingung verschärfen zu $|\{w \in V : (v, w) \in \mathcal{R}\}| \in \{0, 2\}$ für alle $v \in V(G)$, so ist G bzw. T ein **vollständiger Binärbaum**.

Etwas informeller sind Binärbaum also dadurch charakterisiert, dass es eine Relation $\mathcal{R} \subset V \times V$ wie oben gibt, so dass jeder Knoten höchstens zwei Nachfolger hat. Für vollständige Binärbäume lässt sich dies verschärfen dazu, dass jeder Knoten genau zwei Nachfolger hat oder ein Blatt ist, also per Definition keine Nachfolger hat.

2.4 Graphisomorphie, Matchings, Cliquen und stabile Mengen

Es seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ endliche, schlichte Graphen. G_1 und G_2 heißen **isomorph**, kurz $G_1 \cong G_2$, wenn es eine Bijektion $\alpha : V_1 \to V_2$ gibt mit:

$$\{v, w\} \in E_1 \iff \{\alpha(v), \alpha(w)\} \in E_2$$

Entsprechend heißt die Frage, ob zwei Graphen isomorph sind oder nicht, das **Graphisomorphieproblem**. Für weitere Details sei hier an die Literatur, wie etwa Köbler u. a. [1993], verwiesen.

Anzumerken ist hier nur, dass sich bestimmte Aussagen für Baumweite und Boolsche Weite von Graphen zu Teilgraphen (und umgekehrt) übertragen lassen, wie sich noch zeigen wird. Sobald also Schranken für die entsprechenden Größen auf diese Weise bestimmt werden sollen, taucht daher sofort das Graphisomorphieproblem auf. Ebenfalls tritt das Graphisomorphieproblem bei der Suche nach folgenden drei Größen in Erscheinung, wenn auch in stark abgeschwächter Form.

Ist $M \subseteq E(G)$ eines Graphen G mit der Eigenschaft, dass für $e_1, e_2 \in M$ stets $e_1 \cap e_2 = \emptyset$ gilt, so heißt M ein **Matching** in G. Die Kardinalität eines größten Matchings in G heißt **Matchingzahl**, wird mit $\nu(G)$ bezeichnet und ist polynomial bestimmbar.

Gilt mit $|V| = n$ und $|E| = \binom{n}{2} = \frac{n \cdot (n-1)}{2}$, so heißt G ein **vollständiger Graph** bzw. eine **Clique** auf n Knoten und wird K_n bezeichnet. Offensichtlich ist G damit bis auf Isomorphie eindeutig. Die Größe

$$\omega(G) := \max\{\, r \in \{1, \ldots, |V|\} \mid \exists H \leq G : H \cong K_r \,\}$$

heißt **Cliquenzahl** von G und ist $\mathcal{NP}$-schwer zu bestimmen. Eine eng mit der Cliquenzahl verwandte Größe ist die **stabile Mengenzahl**:

$$\alpha(G) := \max\{\, r \in \{1, \ldots, |V|\} \mid \exists H \leq G : H \cong \overline{K_r} \,\}$$

Es gilt offensichtlich $\alpha(G) = \omega(\overline{G})$, so dass das Bestimmen von $\alpha(G)$ ebenfalls $\mathcal{NP}$-schwer ist.

3 Die Begriffe Baumweite, Cliquenweite & Boolsche Weite

In diesem Kapitel sollen die Begriffe Baumweite, Cliquenweite und Boolsche Weite formal definiert und vorgestellt werden. Weitere Ziele sind, Schranken für eine der Größen auf Gültigkeit für andere von ihnen zu untersuchen, um so sowohl die Vorteile als auch die Nachteile der einzelnen Parameter zu beleuchten.

3.1 Baumweite

Zunächst soll nun der Begriff der Baumweite erläutert und vorgestellt werden. Wie zuvor ist $G = (V, E)$ dabei immer ein schlichter, ungerichteter und endlicher Graph. Ziel ist es dabei lediglich die wichtigsten Eigenschaften und Aussagen vorzustellen, ausführlichere Informationen und viele weitere Details sind in der Literatur wie Bodlaender [1998] zu finden.

Definition 3.1.1 *Eine **Baumzerlegung** eines Graphen $G = (V, E)$ ist ein Tupel $(B_{i,i\in I}, T)$, wobei $B_i \subseteq V$ für alle $i \in I$ und $T = (I, F)$ ein Baum mit folgenden Eigenschaften ist:*

1. $\bigcup_{i\in I} B_i = V$
2. *Für alle $\{v, w\} \in E$ existiert ein $i \in I$ mit $v, w \in B_i$*
3. *Für jedes $v \in V$ ist die Familie $\{i \in I : v \in B_i\}$ ein zusammenhängender Teilbaum von T*

*Für eine gegebene Baumzerlegung $(B_{i,i\in I}, T)$ ist die **Weite der Baumzerlegung** definiert als:*

$$tw(B_{i,i\in I}, T) := \max_{i\in I} |B_i| - 1$$

Die Baumweite $tw(G)$ eines Graphen G ist die minimale Weite einer Baumzerlegung, gebildet über alle möglichen Baumzerlegungen von G.

Wohlbekannt ist folgende Anmerkung, die insbesondere die Baumweite als Kriterium für Baum-Ähnlichkeit von Graphen formalisiert:

Bemerkung 3.1.2 *G ist ein genau dann ein Wald, wenn $tw(G) = 1$ gilt.*

Ist also G zusammenhängend, so ist G genau dann ein Baum, falls G Baumweite $tw(G) = 1$ hat. Abschließend sollen nun noch einige grundlegende Aussagen zu Baumzerlegungen vorgestellt werden (nach Bodlaender und Koster [2011]). Die entsprechenden Beweise, bzw. Verweise auf diese, sind ebenfalls dort zu finden.

Lemma 3.1.3 *Die dritte Bedingung an eine Baumzerlegung ist äquivalent zu folgendem Kriterium: Ist j auf dem Pfad von i zu k in T, so folgt $B_i \cap B_k \subseteq B_j$ für alle $i, j, k \in I$.*

Interessant ist es nun herauszufinden, wie der genaue Wert $tw(G)$ für einen gegebenen Graphen ist. Dieses Problem ist allerdings leider $\mathcal{NP}$-schwer, siehe dazu Arnborg und Proskurowski [1986]. Entsprechend sind allerdings Schranken an diese Größe interessant, doch dazu später mehr.

Sofort aus der Definition folgt $tw(G) \geq 1$, falls $E(G) \neq \emptyset$. Weiterhin ist für jeden Graphen G durch $B_1 := V(G)$ und $T := (B_1, \emptyset)$ eine mögliche Baumzerlegung mit Weite $|V(G)| - 1$ gegeben, es gilt also:

Folgerung 3.1.4 *Für jeden Graphen G mit $E(G) \neq \emptyset$ gilt $1 \leq tw(G) \leq |V(G)| - 1$.*

Diese Schranken sind offensichtlich relativ schlecht. Bessere und effizient zu bestimmende Schranken sind daher wünschenswert und werden später noch ausführlicher behandelt. Die beiden folgenden Lemmata formulieren nun noch zwei weitere Eigenschaften von Baumzerlegungen.

Lemma 3.1.5 *Es sei G ein Graph mit Baumweite $tw(G) \leq k$. Dann besitzt G eine Baumzerlegung $(B_{i,i\in I}, T = (I, F))$ mit Weite $\leq k$, so dass $B_i \not\subseteq B_j$ und $B_j \not\subseteq B_i$ für alle $\{i, j\} \in F$ gilt.*

Lemma 3.1.6 *Die Baumweite eines Graphen G entspricht der maximalen Baumweite seiner zusammenhängenden Komponenten.*

3.2 Cliquenweite

Wie sich später noch zeigen wird, ist es ein großer Nachteile des Konzepts der Baumweite, dass sie durch vollständige Teilgraphen sofort in die Höhe getrieben wird. Die Cliquenweite ist eine Größe, die diesen Nachteil nicht hat und einen anderen Ansatz wählt: Es wird versucht einen Graphen durch bestimmte Operationen zu konstruieren und dabei so wenig wie möglich Labels zu verwenden. Dieses Konzept soll hier kurz vorgestellt werden, eine ausführlichere Einführung ist in Courcelle u. a. [1993], wo es erstmals vorgestellt wurde, sowie in Golumbic und Rotics [2000] zu finden. Aus letzterer Arbeit entstammt auch die folgende Einführung.

Definition 3.2.1 (k-Label und Repräsentation) *Für einen Graphen $G = (V, E)$ heißt eine Abbildung*

$$\tau : V \to \{1, \ldots, k\}$$

*ein k-**Label**. Ist τ ein k-Label zu G, so heißt G auch **gelabelter Graph** oder k-Graph und besitzt dann eine **Repräsentation** $\langle V, E, V_1, \ldots, V_k \rangle$, wobei $V_i := \tau^{-1}(i)$ ist.*

In Definition 3.2.1 wird ausdrücklich nicht verlangt, dass τ surjektiv ist. Entsprechend sind möglicherweise manche $V_i = \emptyset$. Doch nun sollen vier Operationen auf gelabelten Graphen definiert werden.

Definition 3.2.2 (Operation $i(v)$) *Ist $\langle V, E, V_1, \ldots, V_k \rangle$ die Repräsentation eines gelabelten Graphen sowie $v \notin V(G)$ und $i \in \{1, \ldots, k\}$. Die Operation $i(v)$ fügt den Knoten v zu G hinzu und labelt ihn anschließend mit i:*

$$i(v) := \langle V \uplus \{v\}, E, V_1, \ldots, V_{i-1}, V_i \uplus \{v\}, V_{i+1}, \ldots, V_k \rangle$$

Die Operation $i(v)$ ist damit erforderlich, um gelabelte Knoten zu erzeugen. Daher werden nun noch drei Operationen auf gelabelten Graphen definiert, mit deren Hilfe sich Knotenlabels ändern, entsprechend gelabelte Knoten verbinden und disjunkte k-Graphen vereinigen lassen.

Definition 3.2.3 (Operation $G \oplus H$) *Für zwei k-Graphen $G = \langle V, E, V_1, \ldots, V_k \rangle$ und $H = \langle V', E', V'_1, \ldots, V'_k \rangle$ mit $V \cap V' = \emptyset$ (ist dies nicht der Fall, so ersetze V' durch eine zu V disjunkte Kopie) wird mit $G \oplus H$ die disjunkte Vereinigung der beiden k-Graphen wie folgt definiert:*

$$G \oplus H := \langle V \uplus V',\ E \uplus E',\ V_1 \uplus V'_1,\ \ldots,\ V_k \uplus V'_k \rangle$$

Definition 3.2.4 (Operation $\eta_{i,j}(G)$) *Ist $G = \langle V, E, V_1, \ldots, V_k \rangle$ ein k-Graph, so verbindet $\eta_{i,j}(G)$ alle $v \in V_i$ mit allen $w \in V_j$, also:*

$$\eta_{i,j}(G) := \langle V,\ E',\ V_1,\ \ldots,\ V_k \rangle \quad \textit{mit:}$$
$$E' := E \cup \{\ \{v, w\}\ : v \in V_i \wedge w \in V_j\ \}$$

Definition 3.2.5 (Operation $\rho_{i \to j}(G)$) *Mit $\rho_{i \to j}(G)$ wird der k-Graph bezeichnet, der entsteht, wenn in $G = \langle V, E, V_1, \ldots, V_k \rangle$ alle Knoten aus V_i mit j umgelabelt werden. Formal bedeutet das also:*

$$\rho_{i \to j}(G) := \langle V,\ E,\ V'_1,\ \ldots,\ V'_k \rangle \quad \textit{mit:}$$
$$V'_k := \begin{cases} \emptyset & \textit{falls } k = i \\ V_i \uplus V_j & \textit{falls } k = j \\ V_k & \textit{sonst} \end{cases}$$

Jeder Ausdruck, der aus beliebig vielen dieser Operationen gebildet werden kann, heißt ein ***k*-Ausdruck**. Der Graph, der durch einen k-Ausdruck r definiert wird, heißt **Wert des Ausdrucks** und wird mit $val(r)$ bezeichnet.

Definition 3.2.6 (Cliquenweite) *Es sei $G = (V, E)$ ein Graph.*

1. *G heißt durch einen k-Ausdruck **konstruierbar**, wenn es einen k-Ausdruck r gibt mit $val(r) = \langle V, E, V_1, \ldots, V_k \rangle$. G geht also aus $val(r)$ hervor, indem alle Labels entfernt werden.*
2. *Das minimale k, so dass G durch einen k-Ausdruck konstruierbar ist, heißt **Cliquenweite** von G und wird mit $cw(G)$ bezeichnet.*

Eine einfache Möglichkeit, um zu einem gegebenen Graphen $G = (\{v_1, \ldots, v_n\}, E)$ einen k-Ausdruck zu bestimmen ist, die Mengen $V_i := \{v_i\}$ zu definieren und für jede Kante $\{v_i, v_j\}$ die Operation $\eta_{i,j}(G)$ zu verwenden. Da jeder schlichte Graph mit mindestens einer Kante keine Schlingen hat, folgt außerdem $cw(G) \geq 2$. Insgesamt ergibt sich daher:

Folgerung 3.2.7 *Für jeden Graphen G mit $E(G) \neq \emptyset$ gilt $2 \leq cw(G) \leq |V(G)|$.*

Um alle vorangegangen Definitionen etwas zu veranschaulichen hilft das folgende Beispiel.

Beispiel 3.2.8 (Konstruktion von Cliquen durch k-Ausdrücke) Es wird nun untersucht, wie die Cliquenweite von vollständigen Graphen aussieht. Ziel ist daher, ausgehend von der Knotenmenge $V(K_n) = \{v_1, v_2, \ldots, v_n\}$, einen Ausdruck r zu konstruieren mit $val(r) \cong K_n$. Für $n = 1$ ist das denkbar einfach:

$$r_1 := 1(v_1) \implies val(r_1) = (\{v_1\}, \emptyset) \cong K_1$$

Also definiert r_1 einen vollständigen Graphen auf einem Punkt. Um nun induktiv aus einem 2-Ausdruck r_{n-1} mit Wert $val(r_{n-1}) \cong K_{n-1}$ einen Ausdruck r_n mit Wert $val(r_n) \cong K_n$ zu konstruieren, wird folgendes verwendet:

$$r_n := \rho_{2\to 1}\ (\ \eta_{1,2}\ (\ 2(v_n) \oplus r_{n-1}\ )\ )$$

Zunächst wird hier also der Knoten v_n mit 2 gelabelt und zu r_{n-1} hinzugefügt. Anschließend werden alle Knoten mit Label 1 mit allen Knoten mit Label 2 verbunden. Nach Konstruktion sind alle Knoten in r_{n-1} aber mit Label 1 versehen, d.h. $\eta_{1,2}$ verbindet gerade v_n zu allen anderen Knoten $v_1, \ldots, v_{n-1}$. Abschließend wird v_n durch $\rho_{2\to 1}$ ebenfalls mit 1 gelabelt, so dass alle Knoten in r_n das Label 1 haben.

Durch Kombination von Folgerung 3.2.7 und Beispiel 3.2.8 ergibt sich ferner:

Folgerung 3.2.9 *Für alle $n \geq 2$ gilt $cw(K_n) = 2$.*

Die Frage, ob das Bestimmen der Cliquenweite eines gegebenen Graphen in $\mathcal{P}$ oder $\mathcal{NP}$ liegt, ist - im Gegensatz zur $\mathcal{NP}$-Schwere der Baumweite - allerdings bisher offen.

3.3 Boolsche Weite

In Analogie zur Baumweite und Cliquenweite soll nun die Boolsche Weite vorgestellt werden. Hier wird erneut ein ganz anderer Ansatz verfolgt und wichtig ist hier zunächst der Begriff des partiellen und vollständigen Zerlegungsbaumes.

Definition 3.3.1 *Ein **partieller Zerlegungsbaum** eines Graphen $G = (V, E)$ ist ein Tupel (T, δ), wobei T ein vollständiger Binärbaum ist und $\delta : T \to Pot(V)\backslash\{\emptyset\}$ folgenden Eigenschaften genügt:*

1. *Ist r die Wurzel von T, so folgt $\delta(r) = V$.*
2. *Ist $t \in T$ kein Blatt, so gilt für die Söhne s_1 und s_2 von t: $\delta(t) = \delta(s_1) \uplus \delta(s_2)$*

Ein Teilbaum von T mit Wurzel $t \in T$ heißt ***vollständiger Teilzerlegungsbaum****, wenn er $|\delta(t)|$ viele Blätter enthält. Hat T selbst $|V|$ viele Blätter, so heißt er* ***vollständiger Zerlegungsbaum****. Die Menge der partiellen bzw. vollständigen Zerlegungsbäume wird mit $\mathcal{T}_p(G)$ und $\mathcal{T}_f(G)$ bezeichnet.*

Diese Definition setzt implizit voraus, dass in einem partiellen Zerlegungsbaum δ stets injektiv ist. Dies ergibt sofort einige Eigenschaften, die kurz festgehalten werden sollen.

Folgerung 3.3.2 *Es gelten die folgenden Eigenschaften:*

1. *Für jeden partiellen Zerlegungsbaum (T, δ) von $G = (V, E)$ gilt: Ist $\mathcal{L} = \mathcal{L}(T)$ die Menge der Blätter von T, dann bildet $\{\delta(\ell) : \ell \in \mathcal{L}\}$ eine Partition von V.*
2. *In einem vollständigen Zerlegungsbaum existiert für jedes $v \in V$ ein eindeutiges Blatt $\ell \in \mathcal{L}$ mit $\delta(\ell) = \{v\}$.*
3. *Ist ein vollständiger Teilzerlegungsbaum mit Wurzel r gegeben, so gibt es für jedes $v \in \delta(r)$ ein eindeutiges Blatt $\ell \in \mathcal{L}$ mit $\delta(\ell) = \{v\}$.*

Jeder Knoten eines Zerlegungsbaumes (T, δ) induziert nun einen Schnitt im Ausgangsgraphen G: Ist $t \in T$, so bildet $(\delta(t), \overline{\delta(t)})$ eine Partition von $V(G)$. In diesem Kontext ist die folgende Definition zu verstehen.

Definition 3.3.3 *Sei $(A, \overline{A})$ ein Schnitt im Graphen G. Die Menge $UN(A)$ von* ***vereinigten Nachbarschaften zu Teilmengen*** *von A ist definiert als:*

$$UN(A) := \{N(X) \cap \overline{A} : X \subseteq A\} = \bigcup_{X \subseteq A} \{N(X) \cap \overline{A}\}$$

Zur Veranschaulichung von Definition 3.3.3 sind in Abbildung 3.1 zwei Schnitte $(A, \overline{A})$ und $(B, \overline{B})$ in einem (4×4)-Gittergraphen dargestellt. Hier ist leicht zu sehen, dass jede Teilmenge von $\overline{A} \cap N(A) = \{3, 7, 11, 15\}$ Nachbarschaft einer geeigneten Teilmenge von A ist, da die Schnittkanten ein perfektes Matching induzieren. Damit folgt hier also $|UN(A)| = 16$. Im Vergleich dazu ist der Schnitt $(B, \overline{B})$ wesentlich geschickter konstruiert, da hier einerseits $\overline{B} \cap N(B)$ nur drei Knoten enthält und andererseits auch nicht jede Teilmenge von $\overline{B} \cap N(B)$ Nachbarschaft einer Menge $X \subseteq B$ ist. Eine genauere Analyse zeigt, dass hier

$$UN(B) = \{\emptyset, \{8\}, \{14\}, \{8, 11\}, \{8, 14\}, \{11, 14\}, \{8, 11, 14\}\}$$

gilt, womit $|UN(B)| = 7$ folgt.

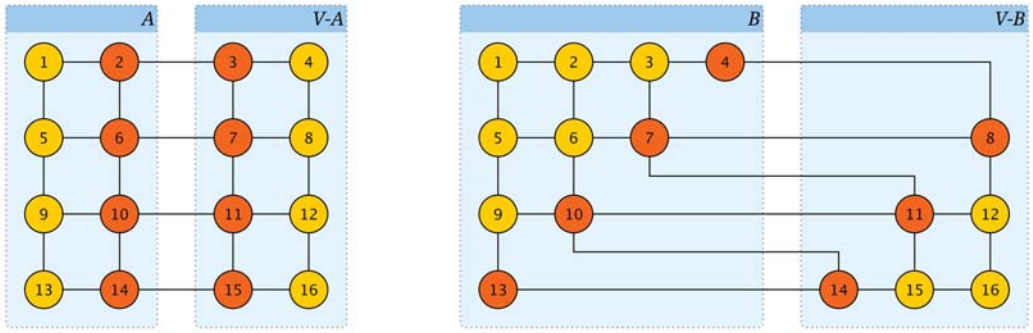

Abbildung 3.1: Zwei Schnitte $(A, \overline{A})$ und $(B, \overline{B})$ in einem (4×4)-Gittergraphen.

Unter Berücksichtigung der Einsichten dieses Beispiels soll nun die Boolsche Weite eines vollständigen Zerlegungsbaumes (T, δ), sowie der eines Graphen G, definiert werden.

Definition 3.3.4 *Die **Boolsche Weite eines partiellen Zerlegungsbaumes** ist definiert als*

$$boolw(T, \delta) := \max_{t \in T} \log_2 |UN(\delta(t))|$$

*und die **Boolsche Weite eines Graphen** ist die minimale Boolsche Weite, gebildet über alle vollständigen Zerlegungsbäume:*

$$boolw(G) := \min_{(T,\delta) \in \mathcal{T}_f(G)} boolw(T, \delta)$$

Zu den Definitionen 3.3.3 und 3.3.4 ist anzumerken, dass für jedes $A \subseteq V(G)$ stets $\emptyset \in UN(A)$ gilt, da einerseits $\emptyset \subseteq A$ und andererseits $\emptyset = N(\emptyset) \cap \overline{A}$ gilt. Es folgt daher $1 \leq |UN(A)| \leq 2^{|V(G)|}$, denn jedes $X \in UN(A)$ erfüllt nach Definition insbesondere $X \subseteq V(G)$.

Da in der Definition der Boolschen Weite der Logarithmus zur Basis 2 gewählt wird, ergibt sich die folgende Aussage, die in Analogie zu den Folgerungen 3.1.4 und 3.2.7 zu sehen ist.

Folgerung 3.3.5 *Für jeden Graphen G mit $E(G) \neq \emptyset$ gilt $1 \leq boolw(G) \leq |V(G)|$.*

Allerdings ist der Logarithmus natürlich unerheblich um einen bzw. den minimerenden vollständigen Zerlegungsbaum zu finden. Dennoch sind diese Schranken an die Boolsche Weite rein elementar und entsprechend schlecht, weshalb es nun primär darum geht, bessere Schranken an die Boolsche Weite zu finden. Ansatzpunkt der Untersuchungen sollen jeweils bekannte, gültige untere Schranken an die Bamweite sein.

3.4 Aussagen für Baumweite im Vergleich zur Boolschen Weite

Ziel ist es nun, bekannte Aussagen über und Schranken an Baumweite vorzustellen und zu analysieren, in weit sie Gültigkeit für Boolsche Weite haben oder sich in leicht anderen Formulierungen dort verallgemeinern lassen. Da die Cliquenweite einer Familie von Graphen genau dann beschränkt ist (vgl. Bui-Xuan u. a. [2009]), wenn es die Boolsche Weite ist, lassen sich große Teile der Ergebnisse im Nachhinein sogar relativ einfach auf Cliquenweite ausweiten.

Obere Schranken an die Baumweite eines Graphen sollen dabei weitestgehend ausgeblendet werden. Da $boolw(G) \leq tw(G)$ für jeden Graphen G gilt, ist trivialerweise jede obere Schranke an die Baumweite auch eine obere Schranke an die Boolsche Weite eines Graphen. Diverse Arbeiten wie Bodlaender und Koster [2010] oder Bouchitté u. a. [2004] beschäftigen sich mit oberen Schranken an die Baumweite, entsprechend lassen sich alle diese Aussagen auf Boolsche Weite übertragen.

Allerdings gilt dies nicht ebenso für Cliquenweite. Um hier gültige obere Schranken zu erhalten, kann mit der Aussage $cw(G) \leq 2^{tw(G)+1} + 1$ aus Courcelle und Olariu [1997] allerdings auf höchstens exponentielle Größe geschlossen werden. Doch dies sei nur aus der Gründen der Vollständigkeit angemerkt.

Mit den vorangehenden Einsichten stehen deswegen in erster Linie untere Schranken an die Baumweite eines Graphen im Zentrum der Analyse. Da diverse Arbeiten über untere Schranken an die Baumweite eines Graphen zur Verfügung stehen, werden im Folgenden einige interessante ausgewählt und vorgestellt. Ausführlichere Informationen sind in eben diesen Arbeiten Bodlaender und Koster [2011], Ramachandramurthi [1995], Sunil Chandran und Subramanian [2003], Sunil Chandran und Subramanian [2005] und Lucena [2003] zu finden.

Als ein erstes Ergebnis sagt Lemma 3.1.6 aus, dass die Baumweite eines Graphen mit der maximalen Baumweite seiner zusammenhängenden Komponenten übereinstimmt, sich eine Baumzerlegung in diesem Sinne auf Teilstrukturen vererbt. In wie weit dieser Umstand, der natürlich später auch noch ausführlicher dargelegt wird, auch Gültigkeit für Boolsche Weite hat, ist dabei erstes Ziel. Dafür ist zunächst der in Listing 3.1 dargestellte Algorithmus nützlich. Es ist relativ schnell zu zeigen, dass dieser korrekt arbeitet.

Listing 3.1: Algorithmus prune

```
1  prune(T, δ, A) {
2     /* aus (T,δ) wird ein vollständiger Zerlegungsbaum (T',δ') zu G[A] berechnet */
3
4     for i ∈ T do
5        δ'(i) := δ(i) ∩ A;
6
7     for i ∈ T do
8        if (δ'(i) == ∅)
9           /* entferne i aus T */
10          T := T - i;
11
12    /* jetzt enthält T keine Knoten j mit δ'(j) = ∅ mehr */
13
14    for i ∈ T do
15       if (|sons(i)|==1)
16          /* jetzt ist i ein Knoten mit nur einem Sohn j, der δ'(i) = δ'(j) erfüllt */
17          /* ersetze i mit seinem Sohn */
18          j := sons(i)[1];
19          sons(i) := sons(j);
20          T := T - j;
21
22    return (T,δ');
23 }
```

Lemma 3.4.1 *Algorithmus 3.1 bestimmt für jedes $\emptyset \neq A \subseteq V(G)$ einen vollständigen Zerlegungsbaum für $G[A]$.*

Beweis

Mit (T', δ') als Output des Algorithmus 3.1 ist nach Definition 3.3.1 zu zeigen:

1. T' ist ein vollständiger Binärbaum.
2. δ' ist wohldefiniert, d.h. es gibt kein $j \in T'$ mit $\delta'(j) = \emptyset$
3. Ist r die Wurzel von T', so folgt $\delta'(r) = A$.
4. Ist $t \in T'$ kein Blatt, so gilt für die Söhne s_1 und s_2 von T: $\delta'(t) = \delta'(s_1) \uplus \delta'(s_2)$

Nach Voraussetzung ist T ein vollständiger Binärbaum, also ist es auch T', bevor ein Knoten gelöscht wird. In der ersten Schleife geht diese Bedingung zwangsläufig verloren, da hier Knoten entstehen, die nur einen Sohn haben. Ist i_0 allerdings ein solcher, so folgt $\delta'(i_0) = \emptyset$ und damit $A \subseteq \delta(i_0)$, da (T, δ) Eigenschaft 3.) genügt. Damit wird i_0 in der zweiten Schleife durch seinen (eindeutigen) Sohn ersetzt, also hat jeder Knoten, der kein Blatt ist, am Ende genau zwei Söhne.

Ferner gelten 2.), da alle solche Knoten nach obigem Verfahren gelöscht werden, und 3.) nach Definition, weil

$$\delta'(r) = \delta(r) \cap A = V \cap A = A$$

nach Voraussetzung an (T, δ) gilt. Abschließend bleibt 4.) zu zeigen. Dazu sei $t' \in T'$ ein beliebiger Knoten, der kein Blatt ist. Nach Konstruktion existiert dann ein $t \in T$ mit $\delta(t) \cap A = \delta'(t)$, so dass kein Sohn von t dies ebenfalls erfüllt. Es bezeichne $s_1 \in T$ und $s_2 \in T$ die Söhne von t sowie $s_1' \in T'$ und $s_2' \in T'$ die Söhne von t'. Dann muss - eine entsprechende Wahl der Indizes vorausgesetzt - nach Konstruktion $\delta(s_1) \cap A = \delta'(s_1')$ sowie $\delta(s_2) \cap A = \delta'(s_2')$ gelten. Daraus folgt

$$\delta'(s_1') \uplus \delta'(s_2') = (\delta(s_1) \uplus \delta(s_2)) \cap A = \delta(t) \cap A = \delta'(t'),$$

was 4.) impliziert und insgesamt die Behauptung folgt. □

Damit kann nun Lemma 3.1.6 auf Boolsche Weite verallgemeinert werden.

Lemma 3.4.2 *Es sei G ein Graph und $H \leq G$ sei eine Zusammenhangskomponente von G. Dann gilt $boolw(H) \leq boolw(G)$.*

Beweis
Da H eine Zusammenhangskomponente ist, gilt $H = G[V(H)]$. Wähle nun einen vollständigen Zerlegungsbaum (T, δ) für G und bestimmme daraus, unter Verwendung von Algorithmus 3.1, einen vollständigen Zerlegungsbaum (T', δ') für H.

Zu zeigen bleibt nun $boolw(T', \delta') \leq boolw(T, \delta)$. Dazu sei $t' \in T'$ beliebig und es wird gezeigt, dass es ein $t \in T$ gibt mit $|UN(t)| \geq |UN(t')|$.

Wähle $t \in T$ so, wie im Beweis von Lemma 3.4.1 - es gibt nach Konstruktion ein $t \in T$ mit $\delta(t) \cap V(H) = \delta'(t)$ und kein Sohn von t erfüllt dies. Dann gilt:

$$\begin{aligned} |UN(\delta(t))| = |\{N(X) \cap \overline{\delta(t)} : X \subseteq \delta(t)\}| &\geq |\{N(X) \cap \overline{\delta(t)} : X \subseteq \delta(t) \cap V(H)\}| \\ &= |\{N(X) \cap \overline{\delta(t)} : X \subseteq \delta'(t')\}| \end{aligned}$$

Nach Voraussetzung ist H eine Zusammenhangskomponente von G, d.h. für jedes $X \subseteq \delta(t) \cap V(H) \subseteq V(H)$ gilt, dass $N(X) \cap \overline{\delta(t)} = N(X) \cap \overline{\delta(t) \cap V(H)}$ ist. Damit folgt:

$$\begin{aligned} |UN(\delta(t))| \geq |\{N(X) \cap \overline{\delta(t)} : X \subseteq \delta'(t')\}| &= |\{N(X) \cap \overline{\delta(t) \cap V(H)} : X \subseteq \delta'(t')\}| \\ = |\{N(X) \cap \overline{\delta'(t')} : X \subseteq \delta'(t')\}| &= |UN(\delta'(t'))| \end{aligned}$$

Das ist die Behauptung. □

Jetzt soll noch untersucht werden, ob für die maximale Boolsche Weite einer Komponente in Lemma 3.4.2 auch Gleichheit gilt. Dafür sei G ein Graph mit Zusammen-

hangskomponenten $G_1, G_2, \ldots, G_d$ und vollständigen Zerlegungsbäumen (T_i, δ_i) für jedes G_i, $1 \leq i \leq d$. Anschließend wird der nun folgende Algorithmus angewendet:

Listing 3.2: Algorithmus `union`

```
union([T_1, T_2, ..., T_d], [δ_1, δ_2, ..., δ_d]) {
   /* es wird ein vollständiger Zerlegungsbaum berechnet,
    * der aus der Vereinigung der (T_i, δ_i) besteht
    */

   if (d == 1) /* Ende der Rekursion */
      return (T_1, δ_1);

   else /* vereinige (T_1, δ_1) und (T_2, δ_2) zu (T, δ) */
      root(T) := r;
      sons(r) := [root(T_1), root(T_2)];
      δ(r) := δ_1(root(T_1)) ⊎ δ_2(root(T_2));

      /* setze δ korrekt */
      for t_1 ∈ T_1 do
         δ(t_1) := δ_1(t_1);

      for t_2 ∈ T_2 do
         δ(t_2) := δ_2(t_2);

      /* arbeite rekursiv auf ([T, T_3, ..., T_d], [δ, δ_3, ..., δ_d]) */
      return union([T, ..., T_d], [δ, ..., δ_d]);
}
```

Da die G_i Zusammenhangskomponenten von G bilden, ist es offensichtlich, dass $UN(r) = \{\emptyset\}$ für alle Knoten r von T gilt, die durch Algorithmus 3.2 konstruiert werden. Die Korrektheit des Algorithmus folgt damit sofort per trivialer Induktion. Festzuhalten bleibt:

Folgerung 3.4.3 *Es sei G ein Graph mit Zusammenhangskomponenten $G_1, \ldots, G_d$. Dann gilt:*

- *Algorithmus 3.2 konstruiert aus vollständigen Zerlegungsbäumen für die G_i einen zulässigen vollständigen Zerlegungsbaum für G, arbeitet also korrekt.*
- *Insbesondere gilt: $boolw(G) = \max\limits_{1 \leq i \leq d} boolw(G_i)$*

Mit Punkt 2.) ist Lemma 3.1.6 also in gleicher Art für Boolsche Weite gültig. Wie sich noch zeigen wird, lassen sich leider viele Aussagen, die für Baumweite gelten, allerdings nicht ebenso auf Boolsche Weite verallgemeinern. Doch dazu später mehr, nun folgt zunächst ein grundsätzliches Ergebnis, das in Analogie zu Beispiel 3.2.8 die Boolsche Weite von vollständigen Graphen betrifft.

Satz 3.4.4 *Für alle $n \in \mathbb{N}$, $n \geq 2$ gilt $boolw(K_n) = \log_2(2) = 1$.*

Beweis
Zunächst liefert Folgerung 3.3.5, dass jeder vollständige Zerlegungsbaum (T, δ) notwendigerweise $boolw(T, \delta) \geq 1$ erfüllt.

Nun ist noch zu zeigen, dass es mindestens einen vollständigen Zerlegungsbaum (T, δ) gibt mit $boolw(T, \delta) = 1$. Dazu sei o.B.d.A. $V(G) = \{v_1, \ldots, v_n\}$ und konstruiere T nach folgenden Regeln:

1. Beginne mit einem Wurzelknoten $r \in T$ mit $\delta(r) = \{v_1, \ldots, v_n\}$.
2. Ist $t \in T$ mit $|\delta(t)| \geq 2$ (also t ist kein Blatt), so bestimme den Knotenindex $j = \min\{1 \leq i \leq n : v_i \in \delta(t)\}$ und füge zu t zwei Söhne s_1 und s_2 hinzu, deren Labels als $\delta(s_1) := \{v_j\}$ und $\delta(s_2) := \delta(t) \backslash \{v_j\}$ definiert werden.
3. Wende 2.) auf s_2 an.

Zeige nun: Für alle $t \in V(T)$ gilt $|UN(\delta(t))| \leq 2$. Es sei dazu $t \in V(T)$ mit $A = \delta(t)$ beliebig. Nach Definition gilt:

$$UN(A) = \{N(X) \cap \overline{A} : X \subseteq A\}$$

Unterscheide hier zwei Fälle: Ist $X = \emptyset$, so folgt $N(X) \cap \overline{A} = \emptyset$. Andernfalls gilt $X \neq \emptyset$, also $N(X) = \overline{X}$ womit $N(X) \cap \overline{A} = \overline{X} \cap \overline{A} = \overline{A}$ folgt. Also gilt für $A \neq V(G)$, dass $UN(A) = \{\emptyset, \overline{A}\}$ ist und es folgt $boolw(T, \delta) = \log_2(2) = 1$. Insgesamt also $boolw(K_n) = 1$. □

Mit diesen Aussagen ist es nun relativ einfach zu zeigen, dass die meisten unteren Schranken an die Baumweite eines Graphens keine Gültigkeit für seine Boolsche Weite haben. Doch dazu sollen nun die Schranken für die Bauweite kurz vorgestellt und dann im Detail untersucht werden.

3.4.1 Grad-basierte Schranken

Es gibt einige bekannte Schranken für Baumweite, die auf Knotengraden basieren. Ausführlicheres dazu ist in Bodlaender und Koster [2011], Ramachandramurthi [1995], Ramachandramurthi [1997] und Sunil Chandran und Subramanian [2005] zu finden, hier sollen nur die Hauptergebnisse kurz vorgestellt werden. Begonnen werden soll mit einer wohlbekannten Aussage, die die Baumweite von vollständigen Graphen beschreibt.

Lemma 3.4.5 *Ist $G = (V, E)$ ein Graph, $H \leq G$ ein vollständiger Teilgraph von G und $(B_{i,i\in I}, T)$ eine zulässige Baumzerlegung für G. Dann gibt es ein $i \in I$ mit $V(H) \subseteq B_i$.*

Als unmittelbare Folgerung ergibt sich mit Lemma 3.4.5 aus der Definition der Baumweite:

Folgerung 3.4.6 *Für alle $n \in \mathbb{N}$, $n \geq 2$ gilt $tw(K_n) = n - 1$.*

Im Kontrast hierzu soll an die Boolsche Weite $boolw(K_n) = 2$ und an die Cliquenweite $cw(K_n) = 2$ von vollständigen Graphen aus Satz 3.4.4 bzw. Folgerung 3.2.9 erinnert werden. Anhand dieser zeigt sich schließlich, dass viele untere Schranken, die für Baumweite gültig sind, für Boolsche Weite nicht gelten. Dennoch sollen die jeweiligen Schranken kurz vorgestellt werden. Begonnen wird mit einer kurzen Definition.

Definition 3.4.7 *Für jeden Graph G sei $\gamma_R(G)$ der Wert:*

$$\gamma_R(G) := \begin{cases} |V(G)| - 1 & \text{falls } G \cong K_{|V(G)|} \\ \min\limits_{\substack{v,w\in V(G):\\ \{v,w\}\notin E(G)}} \max\{d_G(v),\ d_G(w)\} & \text{sonst} \end{cases}$$

Wird $\delta_2(G) := \min\{d_G(v) \mid v \in V(G) \ \wedge \ \exists w \in V(G) : d_G(w) \leq d_G(v)\}$ als der zweit-kleinste Knotengrad in G definiert, ergeben sich damit die folgenden drei, in Linearzeit zu bestimmenden, unteren Schranken an die Baumweite von G.

Lemma 3.4.8 (Ramachandramurthi [1995]) *Für jeden Graphen G gilt:*

$$tw(G) \geq \gamma_R(G) \geq \delta_2(G) \geq \delta(G)$$

Durch die grundsätzlich einfache Bestimmbarkeit sind diese Schranken auch durchaus als praktisch relevant einzustufen. Leider haben alle diese Größen keine Gültigkeit für die Boolsche Weite.

Lemma 3.4.9 *Es gibt eine unendliche Familie H_k von Graphen, so dass in Lemma 3.4.8 alle Schranken mit Gleichheit erfüllt werden, genauer*

$$tw(H_k) = \gamma_R(H_k) = \delta_2(H_k) = \delta(H_k) = k - 1 \ \in \ \Theta(|V(H_k)|)$$

und ferner $boolw(H_k) \in \mathcal{O}(1)$ für alle k gilt. Insbesondere lässt sich Lemma 3.4.8 nicht auf Boolsche Weite ausweiten.

Beweis
Es sei für alle $k \in \mathbb{N}$, $k \geq 2$ der Graph H_k der vollständige Graph auf k Knoten: $H_k := K_k$. Dann gilt nach Definition $\gamma_R(H_k) = \delta_2(H_k) = \delta(H_k) = k - 1$ sowie $tw(H_k) = k - 1$ nach Folgerung 3.4.6. Ferner gilt $boolw(H_k) = 1 \in \mathcal{O}(1)$ nach Satz 3.4.4. Das ist die Behauptung. □

Eine weitere untere Schranke an die Baumweite beruht auf der Taillenweite und auf dem durchschnittlichen Knotengrad. Näheres dazu ist in Bodlaender und Koster [2011] und Sunil Chandran und Subramanian [2005] zu finden, hier soll nur folgendes Lemma zitiert werden.

Lemma 3.4.10 (Sunil Chandran und Subramanian [2005]) *Es sei G ein Graph mit Knotenmenge $V = V(G)$, Taillenweite $g = g(G)$ und durchschnittlichem Knotengrad $\overline{d} = \frac{1}{|V|} \sum_{v \in V} d_G(v)$. Dann gilt:*

$$tw(G) \geq \max \left\{ \left\lceil \frac{\overline{d}}{2} \right\rceil, \frac{1}{4 \cdot e \cdot (g+1)} \cdot \left(\overline{d} - 1\right)^{\left\lfloor \frac{g-1}{2} \right\rfloor} - 2 \right\}$$

Im Ausdruck auf der rechten Seite der Ungleichung bezeichnet e die Eulersche Zahl $e = \exp(1)$ und der durchschnittliche Knotengrad kann über die wohlbekannte Identität $\overline{d} = \frac{1}{|V|} \sum_{v \in V} d(v) = \frac{2 \cdot |E|}{|V|}$ effizienter bestimmt werden. Die Taillenweite kann auch in polynomialer Zeit bestimmt werden, d.h. die Schranke kann insgesamt ebenfalls in Polynomialzeit berechnet werden, wenngleich dies hier nicht weiter wichtig ist. Auch hier ergibt sich wieder ein negatives Ergebnis, was die Gültigkeit der Schranke für die Boolsche Weite anbelangt.

Lemma 3.4.11 *Es gibt eine unendliche Familie H_k von Graphen, so dass mit der Notation aus Lemma 3.4.10*

$$tw(H_k) \in \Theta(|V(H_k)|) \qquad \text{sowie}$$
$$\min \left\{ \left\lceil \frac{\overline{d}}{2} \right\rceil, \frac{1}{4 \cdot e \cdot (g+1)} \cdot \left(\overline{d} - 1\right)^{\left\lfloor \frac{g-1}{2} \right\rfloor} - 2 \right\} \in \Theta(|V(H_k)|)$$

gilt und ferner $boolw(H_k) \in \mathcal{O}(1)$ für alle k ist.

Beweis
Wie im Beweis von Lemma 3.4.9 sei für alle $k \in \mathbb{N}$, $k \geq 3$ der Graph $H_k := K_k$ der vollständige Graph der Ordnung k.

Nach Satz 3.4.4 gilt auch hier $boolw(H_k) \in \mathcal{O}(1)$, so dass der erste Teil zu zeigen bleibt. Die beiden Terme vereinfachen sich jeweils zu

$$\left\lceil \frac{\overline{d}}{2} \right\rceil = \left\lceil \frac{2 \cdot |E(H_k)|}{2 \cdot |V(H_k)|} \right\rceil = \left\lceil \frac{|V(H_k)| \cdot (|V(H_k)| - 1)}{|V(H_k)|} \right\rceil = |V(H_k)| - 1 \in \Theta(|V(H_k)|)$$

sowie unter Verwendung von $g(K_k) = 3$ zu:

$$\begin{aligned} \frac{1}{4 \cdot e \cdot (g+1)} \cdot \left(\overline{d} - 1\right)^{\lfloor \frac{g-1}{2} \rfloor} - 2 &= \frac{1}{16 \cdot e} \cdot (2 \cdot (|V(H_k)| - 1) - 1) - 2 \\ &= \frac{|V(H_k)|}{8 \cdot e} - \frac{32 \cdot e + 3}{16 \cdot e} \in \Theta(|V(H_k)|) \end{aligned}$$

Insgesamt folgt die Behauptung. □

Alle Aussagen dieses Abschnitts zeigen also, dass Baumweite zwar an den Knotengrad gekoppelt ist, die Boolsche Weite aber weitestgehend unabhängig davon zu sehen ist.

3.4.2 Struktur-basierte Schranken

Die Baumweite eines Graphen ist natürlich eng an die Struktur des Graphen gebunden - je "ähnlicher" die Struktur eines Graphens der eines Baumes ist, desto kleiner ist seine Baumweite. Auch wenn nicht klar definiert ist, was "Ähnlichkeit einer Struktur" ist, hat dies doch einige Konsequenzen: Ein Graph vererbt diese Eigenschaften erwartungsgemäß an alle seine Teilgraphen. Dies soll nun einerseits formal ausgeführt und andererseits bei Boolscher Weite untersucht werden. Begonnen wird mit einem einfach zu beweisenden Ergebnis.

Lemma 3.4.12 (Bodlaender und Koster [2011]) *Es sei $G = (V, E)$ ein Graph und $W \subseteq V$ eine Menge von Knoten. Dann gilt $tw(G[W]) \leq tw(G)$.*

Beweis

In $(B_{i,i\in I}, T = (I, F))$ sei eine beliebige Baumzerlegung von G mit Weite k gegeben. Mit $Y_i := B_i \cap W$ für alle $i \in I$ ist $(Y_{i,i\in I}, T = (I, F))$ eine Baumzerlegung von $G[W]$ mit Weite $\leq k$. □

Folgerung 3.4.13 (Bodlaender und Koster [2011]) *Ist H ein Teilgraph von G, so folgt $tw(H) \leq tw(G)$.*

Beweis

Für jedes $E' \subseteq E$ gilt: Ist $(B_{i,i\in I}, T = (I, F))$ eine Baumzerlegung für $G = (V, E)$, so ist es auch eine zulässige Baumzerlegung für $G' = (V, E')$. Die Behauptung folgt mit Lemma 3.4.12. □

Nun wird untersucht, in wie fern sich diese Ergebnisse von Baumweite auf Boolsche Weite übertragen lassen. Offensichtlich ist jeder Graph auf n Punkten Teilgraph eines K_n, es muss also nur eine Graphklasse gefunden werden, deren Boolsche Weite nicht-konstant ist.

Definition 3.4.14 *Es sei $k \in \mathbb{N}$. Der Graph $G_k = (V, E)$ mit*

- $V = \{v_{i,j} : 1 \leq i, j \leq k\}$
- $E = E_1 \uplus E_2$ *wobei*
  - $E_1 = \{\{v_{i,j},\ v_{i,j+1}\} : 1 \leq i \leq k,\ 1 \leq j \leq k-1\}$
  - $E_2 = \{\{v_{i,j},\ v_{i+1,j}\} : 1 \leq i \leq k-1,\ 1 \leq j \leq k\}$

*heißt $(k \times k)$-**Gittergraph**.*

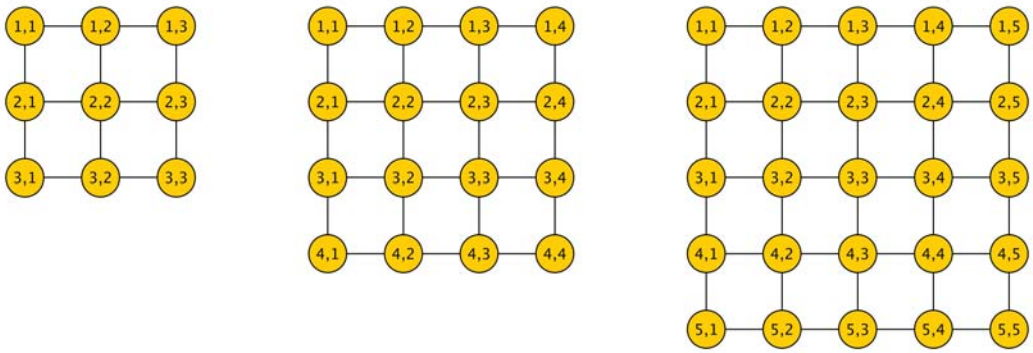

Abbildung 3.2: Die Gittergraphen G_3, G_4 und G_5

Für diese Gittergraphen gilt nach Bui-Xuan u. a. [2009] folgende Aussage:

Satz 3.4.15 (Bui-Xuan u. a. [2009]) *Es sei $k \geq 2$ und G_k der $(k \times k)$-Gittergraph. Dann gilt:*

$$\frac{1}{6} \cdot k \leq boolw(G_k) \leq k + 1$$

Wie der genaue Wert $boolw(G_k)$ ist, ist bisher offen. In Kapitel 4 wird dieses Problem wird noch genauer behandelt. Hier genügt die Eigenschaft, dass $boolw(G_k) \in \Theta(k)$ und $k = \sqrt{|V(G_k)|}$ ist.

Folgerung 3.4.16 *Es gibt zwei unendliche Familien H_k und L_k von Graphen, so dass für alle $k \in \mathbb{N}$, $k \geq 2$ der Graph L_k ein Teilgraph von H_k ist und ferner gilt:*

$$boolw(L_k) \in \Theta\left(\sqrt{|V(L_k)|}\right) \quad \text{sowie} \quad boolw(H_k) \in \mathcal{O}(1)$$

Insbesondere ist die Boolsche Weite eines Teilgraphen keine untere Schranke an die Boolsche Weite des ursprünglichen Graphen.

Beweis
Der Graph L_k sei ein $(k \times k)$-Gitter und H_k sei der vollständige Graph auf k^2 Punkten: $L_k := G_k$ sowie $H_k := K_{k^2}$. Dann ist L_k ein Teilgraph von H_k (bzw. isomorph zu einem) und die Behauptung folgt aus den Sätzen 3.4.4 bzw. 3.4.15. □

Mit der letzten Folgerung scheidet also der Wert eines beliebigen Teilgraphen als untere Schranke für die Boolsche Weite aus. Ist der Teilgraph allerdings induziert, so ergibt sich tatsächlich auch eine gültige untere Schranke. Da jede Komponente eines Graphen insbesondere ein induzierter Teilgraph ist, ist die folgende Aussage außerdem eine Verallgemeinerung von Lemma 3.4.2.

Satz 3.4.17 *Es sei $G = (V, E)$ ein Graph und $W \subseteq V$ eine Menge von Knoten. Dann gilt $boolw(G) \geq boolw(G[W])$.*

Beweis
Wähle einen beliebigen vollständigen Zerlegungsbaum (T, δ) für G. Mit Algorithmus 3.1 ergibt sich ein vollständiger Zerlegungsbaum (T', δ') für $G[W]$. Gezeigt wird nun, dass $boolw(T, \delta) \geq boolw(T', \delta')$ gilt. Dazu sei $t' \in T'$ ein beliebiger Knoten und es wird gezeigt, dass es ein $t \in T$ gibt, so dass $|UN_G(\delta(t))| \geq |UN_{G[W]}(\delta'(t'))|$ ist.

Der Knoten $t \in T$ ist, wie im Beweis der Lemmata 3.4.1 und 3.4.2, eindeutig dadurch bestimmmt, dass $\delta(t) \cap W = \delta'(t')$ ist und kein Sohn von t dies ebenfalls erfüllt. Zu zeigen bleibt, dass auch $|UN_G(\delta(t))| \geq |UN_{G[W]}(\delta'(t'))|$ ist. Es gilt:

$$\begin{aligned} |UN_G(\delta(t))| = |\{N_G(X) \cap \overline{\delta(t)} : X \subseteq \delta(t)\}| &\geq |\{N_G(X) \cap \overline{\delta(t)} : X \subseteq \delta(t) \cap W\}| \\ &= |\{N_G(X) \cap \overline{\delta(t)} : X \subseteq \delta'(t')\}| \end{aligned}$$

Zeige daher, dass $|\{N_G(X) \cap \overline{\delta(t)} : X \subseteq \delta'(t')\}| \geq |\{N_{G[W]}(X) \cap \overline{\delta(t)} : X \subseteq \delta'(t')\}|$ erfüllt ist. Hierfür ist einerseits wichtig, dass jedes mögliche X auf der rechten Seite auch auf der linken Seite gewählt werden kann. Entscheidend für den Schluss ist allerdings folgende Behauptung: Erfüllen die Mengen $X_1, X_2 \subseteq \delta'(t')$ die Bedingung $N_{G[W]}(X_1) \cap \overline{\delta(t)} \neq N_{G[W]}(X_2) \cap \overline{\delta(t)}$, so folgt schon $N_G(X_1) \cap \overline{\delta(t)} \neq N_G(X_2) \cap \overline{\delta(t)}$.

Es gibt also o.B.d.A. ein $x \in N_{G[W]}(X_1) \cap \overline{\delta(t)}$ mit $x \notin N_{G[W]}(X_2) \cap \overline{\delta(t)}$ (sonst vertausche X_1 und X_2). Da $x \in N_{G[W]}(X_1) \cap \overline{\delta(t)}$ ist, gilt insbesondere $x \in W \cap \overline{\delta(t)}$. Also folgt aus $x \notin N_{G[W]}(X_2) \cap \overline{\delta(t)}$, da $G[W]$ ein induzierter Teilgraph ist, auch $x \notin N_G(X_2) \cap \overline{\delta(t)}$ und somit ergibt sich $N_G(X_1) \cap \overline{\delta(t)} \neq N_G(X_2) \cap \overline{\delta(t)}$.

Insgesamt folgt daher:

$$\begin{aligned}
|UN_G(\delta(t))| &\geq |\{N_G(X) \cap \overline{\delta(t)} : X \subseteq \delta'(t')\}| \\
&\geq |\{N_{G[W]}(X) \cap \overline{\delta(t)} : X \subseteq \delta'(t')\}| \\
&= |\{N_{G[W]}(X) \cap \overline{\delta(t)} \ \cup \ \emptyset : X \subseteq \delta'(t')\}| \\
&= |\{N_{G[W]}(X) \cap \overline{\delta(t)} \ \cup \ N_{G[W]}(X) \cap \overline{W} : X \subseteq \delta'(t')\}| \\
&= |\{N_{G[W]}(X) \cap (\overline{\delta(t)} \cup \overline{W}) : X \subseteq \delta'(t')\}| \\
&= |\{N_{G[W]}(X) \cap \overline{\delta(t) \cap W} : X \subseteq \delta'(t')\}| \\
&= |\{N_{G[W]}(X) \cap \overline{\delta'(t')} : X \subseteq \delta'(t')\}| \\
&= |UN_{G[W]}(\delta'(t'))|
\end{aligned}$$

Das ist die Behauptung. □

Damit ist es also gelungen, eine weitere Schranke für Baumweite auf Boolsche Weite zu übertragen. Für die Baumweite gibt es aber auch noch eine verwandte Strategie, nämlich die Baumweite eines Minors. Einerseits gilt das folgende Lemma.

Lemma 3.4.18 (Bodlaender und Koster [2011]) *Der Graph $H = (W, F)$ sei ein Minor von $G = (V, E)$. Dann gilt $tw(H) \leq tw(G)$.*

Andererseits ist jeder Teilgraph von G auch sein Minor, also impliziert Folgerung 3.4.16 sofort, dass die Boolsche Weite eines Minors M von G im Allgemeinen keine untere Schranke an die Boolsche Weite von G ist.

Folgerung 3.4.19 *Es gibt zwei unendliche Familien H_k und L_k von Graphen, so dass für alle $k \in \mathbb{N}$, $k \geq 2$ der Graph L_k ein Minor von H_k ist und ferner gilt:*

$$boolw(L_k) \in \Theta\left(\sqrt{|V(G)|}\right) \quad \text{sowie} \quad boolw(H_k) \in \mathcal{O}(1)$$

Hier soll aber noch etwas mehr gezeigt werden, nämlich dass es auch Graphen gibt, die beliebig große Boolsche Weite haben, und gleichzeitig k Teilgraphen bzw. Minoren beliebiger Ordnung besitzen, deren Baumweite in $\Theta(k)$ liegt, während ihre Boolsche Weite konstant ist. Um diese Aussage zu zeigen sind einige Hilfsmittel notwendig. Daher zunächst ein wichtiger Satz.

Satz 3.4.20 (Bui-Xuan u. a. [2009]) *Für jeden Graphen G gilt:*

$$\log_2(cw(G)) - 1 \leq boolw(G) \leq cw(G)$$

Weiter gibt es für jedes $k \in \mathbb{N}$, $k \geq c$ für eine Konstante c, Familien von Graphen L_k und U_k mit $cw(L_k) \geq k$ sowie $cw(U_k) \geq k$, so dass $boolw(L_k) \leq 2 \cdot \log_2(cw(L_k)) + 4$ und $boolw(U_k) \geq \left\lfloor \frac{1}{6} \cdot cw(U_k) \right\rfloor - 1$ gilt.

Die Schranken aus Satz 3.4.20 sind also scharf bis auf Multiplikation mit Konstanten. Weiterhin induziert damit auch jede untere Schranke an die Cliquenweite eines Graphen auch eine an die Boolsche Weite: Ist $f(G) \leq cw(G)$ für einen Graphen G, so folgt $\log_2(f(G)) - 1 \leq \log_2(cw(G)) - 1 \leq boolw(G)$. Ziel ist es aber nun, über diese Schranken, für jedes $m \in \mathbb{N}$, eine unendliche Graphklasse zu finden, so dass jeder Graph dieser Klasse mindestens m Teilgraphen bzw. Minoren der Ordnung $k \geq m$, mit jeweils konstanter Boolscher Weite und Baumweite $k - 1$, besitzt und andererseits diese Familie selbst nicht-konstante Boolsche Weite hat.

Definition 3.4.21 (Golumbic & Rotics) *Es sei $k \in \mathbb{N}$. Der Graph $HG_k = (V, E)$ definiert durch*

- $V := \{v_{i,j} : 1 \leq i, j \leq k\}$
- $E := E_1 \uplus E_2 \uplus E_3$ *wobei*
  - $E_1 = \{\{v_{i_1,j},\ v_{i_2,j}\} : 1 \leq i_1 \leq k,\ 1 \leq i_2 \leq k,\ i_1 \neq i_2,\ 1 \leq j \leq k\}$
  - $E_2 = \{\{v_{i_1,j},\ v_{i_2,j+1}\} : j \in 2{\cdot}\mathbb{N}{-}1,\ j \leq k{-}1,\ 2 \leq i_1 \leq k,\ 1 \leq i_2 \leq i_1{-}1\}$
  - $E_3 = \{\{v_{i_1,j},\ v_{i_2,j+1}\} : j \in 2{\cdot}\mathbb{N},\ j \leq k{-}1,\ 1 \leq i_1 \leq k{-}1,\ i_1{+}1 \leq i_2 \leq k\}$

*heißt ein **H-Gitter** der Ordnung k^2.*

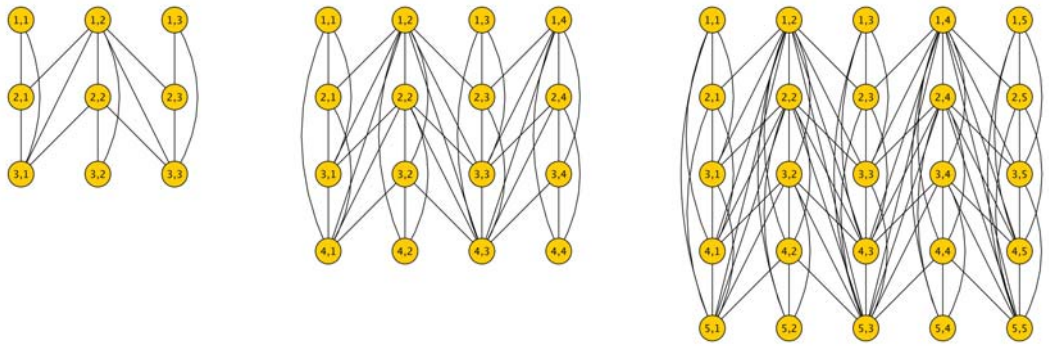

Abbildung 3.3: Die H-Gitter HG_3, HG_4 und HG_5

Die Kanten aus E_1 bedeuten, dass für alle $1 \leq \ell \leq k$ jede Spalte, d.h. die Knotenmenge $S_\ell = \{v_{i,\ell} : 1 \leq i \leq k\}$, eine Clique bildet. Also gilt:

Bemerkung 3.4.22 *K_k ist ein Teilgraph bzw. Minor von HG_k. Weiter besitzt HG_k genau k solcher Teilgraphen.*

Ebenfalls wurde folgende wichtige Aussage bewiesen:

Satz 3.4.23 (Golumbic & Rotics) *Für alle $k \in \mathbb{N}$ gilt $k \leq cw(HG_k) \leq k+1$.*

Es wird vermutet, dass $cw(HG_k) = k+1$ gilt. Allerdings reicht hier folgende Eigenschaft, die sich sofort aus den Sätzen 3.4.23 und 3.4.20 ergibt.

Folgerung 3.4.24 *Für alle $k \in \mathbb{N}$ gilt $boolw(HG_k) \geq \log_2(k) - 1$.*

Damit sind alle Werkzeuge vorhanden, um das eigentliche Hauptergebnis zu formulieren. Dieses folgt jetzt aus den Folgerungen 3.4.6 und 3.4.24, Bemerkung 3.4.22 sowie Satz 3.4.4.

Folgerung 3.4.25 *Es existiert eine unendliche Familie H_k von Graphen mit jeweils $|V(H_k)| = k^2$ Knoten, so dass einerseits $boolw(H_k) \overset{k\to\infty}{\longrightarrow} \infty$ gilt, und andererseits k disjunkte Teilgraphen bzw. Minoren der Ordnung k von H_k existieren, deren Baumweite $k-1$ und Boolsche Weite konstant ist.*

3.4.3 Spektrale Schranken

Alle bisher vorgestellten Schranken für Baumweite - und damit die daraus abzuleiten versuchten Schranken für Boolsche Weite - basierten alle auf der unmittelbaren Struktur des Graphen im Sinne von Teilgraphen oder Knotengraden. Nach Sunil Chandran und Subramanian [2003] gibt es noch eine weitere Schranke, die einen abstrakteren Ansatz wählt. Für diese ist die folgende Definition nötig.

Definition 3.4.26 (Laplace-Matrix) *Für einen Graphen $G = (V, E)$ mit Knotenmenge $V = \{v_1, v_2, \ldots, v_n\}$ heißt die $n \times n$-Matrix L mit*

$$L_{i,j} = \begin{cases} d_G(v_i) & \text{falls } i = j \\ 0 & \text{falls } i \neq j \text{ und } \{v_i, v_j\} \notin E \\ -1 & \text{falls } i \neq j \text{ und } \{v_i, v_j\} \in E \end{cases}$$

*für $1 \leq i, j \leq n$ die **Laplace-Matrix** von G.*

Die Laplace-Matrix ist symmetrisch, hat also ausschließlich reelle Eigenwerte. Mit Hilfe dieser ergibt sich nach Sunil Chandran und Subramanian [2003] die folgende Schranke an die Baumweite eines Graphen.

Satz 3.4.27 (Sunil Chandran und Subramanian [2003]) *Es sei $G = (V, E)$ ein Graph mit Maximalgrad $\Delta = \Delta(G)$ und Laplace-Matrix L, deren zweit-kleinster Eigenwert μ sei. Dann folgt:*

$$tw(G) \geq \left\lfloor \frac{3 \cdot |V|}{4} \cdot \left(\frac{\mu}{\Delta + 2 \cdot \mu} \right) \right\rfloor - 1$$

Nun soll erneut untersucht werten, ob sich diese Schranke auf Boolsche Weite übertragen lässt. Dazu vorab folgendes Lemma.

Lemma 3.4.28 *Die Laplace-Matrix des vollständigen Graphen $L = L(K_n)$ hat genau die Eigenwerte 0 mit Vielfachheit 1 und n mit Vielfachheit $n-1$. Oder äquivalent: Das charakteristische Polynom ist $\chi_L(\lambda) = \lambda \cdot (n - \lambda)^{n-1}$.*

Beweis
Nach Definition der Laplace-Matrix gilt (wobei hier E_n die $n \times n$-Einheitsmatrix bezeichnet):

$$L(K_n) = \begin{pmatrix} n-1 & -1 & \cdots & -1 \\ -1 & n-1 & \cdots & -1 \\ \vdots & \vdots & & \vdots \\ -1 & -1 & \cdots & n-1 \end{pmatrix} = n \cdot E_n - \underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}}_{=:1_n} = n \cdot E_n - 1_n$$

Also gilt:

$$\begin{aligned} \lambda \text{ ist Eigenwert von } L(K_n) &\iff \exists v \in \mathbb{R}^n : L(K_n) \cdot v = \lambda \cdot v \\ &\iff \exists v \in \mathbb{R}^n : (n \cdot E_n - 1_n) \cdot v = \lambda \cdot v \\ &\iff \exists v \in \mathbb{R}^n : n \cdot E_n \cdot v - \lambda \cdot v = 1_n \cdot v \\ &\iff \exists v \in \mathbb{R}^n : (n - \lambda) \cdot v = 1_n \cdot v \end{aligned}$$

Betrachte nun die beiden Fälle: Ist $\lambda = 0$, so folgt $v \in \left\langle (1, 1, \ldots, 1)^{Tr} \right\rangle$. Gilt $\lambda = n$, so ist zwangsläufig $v \in kern(1_n)$. Da $rg(1_n) = 1$ ist, folgt $\dim(kern(1_n)) = n - 1$. Also gibt es keine anderen Möglichkeiten für λ und die Behauptung folgt. □

Mit dieser Aussage kann jetzt das eigentliche Ergebnis gezeigt werden.

Folgerung 3.4.29 *Es gibt eine unendliche Familie H_k von Graphen, so dass mit der Notation aus Satz 3.4.27*

$$\left\lfloor \frac{3 \cdot |V(H_k)|}{4} \cdot \left(\frac{\mu}{\Delta + 2 \cdot \mu} \right) \right\rfloor - 1 \in \Theta(|V(H_k)|)$$

gilt, während $boolw(H_k) \in \mathcal{O}(1)$ *für alle* k *ist. Insbesondere lässt sich das Kriterium nicht auf Boolsche Weite ausweiten.*

Beweis

Wähle erneut $H_k := K_n$ als den vollständigen Graphen auf n Knoten. Nach Satz 3.4.4 bleibt der erste Teil zu zeigen. Mit $k = |V(H_k)|$ gilt dann $\Delta(H_k) = k - 1$ und $\mu = k$ nach Lemma 3.4.28. Insgesamt folgt also:

$$\begin{aligned}
\left\lfloor \frac{3 \cdot |V(H_k)|}{4} \cdot \left(\frac{\mu}{\Delta + 2 \cdot \mu} \right) \right\rfloor - 1 &= \left\lfloor \frac{3}{4} \cdot \frac{k^2}{k - 1 + 2 \cdot k} \right\rfloor - 1 \\
= \left\lfloor \frac{3}{4} \cdot \frac{k^2}{3 \cdot k - 1} \right\rfloor - 1 \qquad &= \left\lfloor \frac{3}{4} \cdot \frac{k}{3 - \frac{1}{k}} \right\rfloor - 1 \\
&= \left\lfloor \frac{1}{4} \cdot \frac{k}{1 - \frac{1}{3 \cdot k}} \right\rfloor - 1 \ \in \Theta(k)
\end{aligned}$$

Das ist die Behauptung. □

Festzuhalten bleibt also, dass auch diese untere Schranke keine Gültigkeit für Boolsche Weite hat.

3.4.4 Maximale Kardinalitätssuche

Ursprünglich zur Erkennung chordaler[1] Graphen wurde in Tarjan und Yannakakis [1984] die Maxiamle Kardinalitätssuche vorgestellt, die nach Lucena [2003] auch verwendet werden kann um eine untere Schranke an die Baumweite zu bestimmen. Die Idee dabei ist (nach Bodlaender und Koster [2011]), die Knoten eines Graphen G in einer bestimmten Reihenfolge zu besuchen, beginnend bei keinem besuchten Knoten und der nächste Knoten, der besucht wird, muss stets maximal viele bereits besuchte Nachbarn haben. Jede solche Anordnung der Knoten heißt **MCS-Ordnung**.

Insbesondere kann also der erste Knoten beliebig gewählt werden, da zu diesem Zeitpunkt gar kein Knoten einen bereits besuchten Nachbarn hat. Damit existieren, für einen festen Graphen, im Allgemeinen insbesondere verschiedene solche Anordnungen. Die hier entscheidende Eigenschaft von MCS-Ordnungen formuliert folgender Satz, für Weiteres dazu siehe Lucena [2002] und Lucena [2003].

Satz 3.4.30 (Lucena [2002, 2003]) *Für jeden Graphen* $G = (V, E)$ *und jede MCS-Ordnung* π *von* G *ist die Baumweite von* G *mindestens:*

$$MCSLB(G, \pi) = \max_{v \in V} |\{\{u, v\} \in E : \pi(u) < \pi(v)\}|$$

[1] Ein Graph heißt chordal, wenn es keine induzierten Kreise der Länge ≥ 4 gibt.

Nach Bodlaender und Koster [2007] ist es für jedes feste $k \geq 7$ $\mathcal{NP}$-vollständig zu entscheiden, ob ein gegebener Graph $G = (V, E)$ eine MCS-Ordnung π besitzt mit $MCSLB(G, \pi) \geq k$. Doch hier genügt das folgende Ergebnis.

Lemma 3.4.31 *Es gibt eine unendliche Familie H_k von Graphen mit $V(H_k) = k$, so dass*

$$MCSLB(H_k, \pi) = tw(H_k) = k - 1$$

ist, während $boolw(H_k) \in \mathcal{O}(1)$ gilt. Insbesondere lässt sich Satz 3.4.30 nicht auf Boolsche Weite übertragen.

Beweis
Wähle $H_k = K_k$ als den vollständigen Graphen auf k Knoten. Nach Satz 3.4.4 gilt $boolw(H_k) \in \mathcal{O}(1)$ und $tw(H_k) = k - 1$ liefert Folgerung 3.4.6. Es bleibt also $MCSLB(H_k, \pi) = k - 1$ zu zeigen. Dazu sei π eine beliebige MCS-Ordnung auf H_k. Es sei v_0 der letzte von π aufgezählte Knoten. Dann gilt:

$$|\{\{u, v_0\} \in E : \pi(u) < \pi(v_0)\}| = |V(H_k)| - 1 = k - 1$$

Mit Satz 3.4.30 folgt $MCSLB(H_k, \pi) = k - 1$. Das ist die Behauptung. □

4 Analyse der Boolschen Weite gewisser Graphklassen

In diesem Kapitel soll die Boolsche Weite von bestimmten perfekten Graphen genauer untersucht werden, deren Baumweite und Cliquenweite in $\Omega(\sqrt{|V|})$ ist. Begonnen wird mit einer kurze Einführung und einem Abschnitt über dominierte Mengen, die ein sehr nützliches Hilfsmittel zur Berechnung von $|UN(A)|$ - und damit für die Boolsche Weite an sich - darstellen.

4.1 Einführung

In Kapitel 3 ist bereits die Klasse G_k der Gittergraphen eingeführt worden, dort diente sie als Hilfsmittel um zu zeigen, dass sich Boolsche Weite nicht notwendigerweise auf Teilgraphen vererbt, vgl. Folgerung 3.4.16. Dort hatte die Abschätzung $\frac{1}{6} \cdot k \leq boolw(G_k) \leq k+1$, oder etwas allgemeiner $boolw(G_k) \in \Theta\left(\sqrt{|V(G_k)|}\right)$, aus Satz 3.4.15 genügt. Wie allerdings die Boolsche Weite genau ist, stellt ein bisher offenes Problem dar (siehe dazu auch Hvidevold u. a. [2011] und Bui-Xuan u. a. [2009]), dem u.a. nun nachgegangen werden soll.

Nach Definition ist dafür $|UN(A)|$ über $A = \delta(t)$, für $t \in T$, zu maximieren, wobei (T, δ) ein optimaler vollständiger Zerlegungsbaum für G_k ist, also wiederum minimal mit dieser Eigenschaft ist. Nach Definition ist $UN(A) = \{N(X) \cap \overline{A} : X \subseteq A\}$, d.h. entscheidend für die Minimierung von $|UN(A)|$ sind folgende Punkte: Wie groß ist $|A|$ mindestens und wie viele $X \subseteq A$ gibt es, für festes A, so dass diese paarweise verschiedene Nachbarschaften in $\overline{A}$ haben?

Ist $a \in A$ mit $N(a) \cap \overline{A} = \emptyset$, so ist a offensichtlich irrelevant für die Minimierung von $|UN(A)|$. Dies soll kurz festgehalten werden.

Bemerkung 4.1.1 *Ist $G = (V, E)$ ein Graph mit $a \in A \subseteq V$ sowie $N(a) \cap \overline{A} = \emptyset$, so gilt:*

$$UN(A) = \{N(X) \cap \overline{A} : X \subseteq A - \{a\}\}$$

Wird Bemerkung 4.1.1 auf alle $a \in A \backslash N(\overline{A})$ angewendet, ergibt sich eine wichtige Vereinfachung für die Minimierung von $|UN(A)|$.

Folgerung 4.1.2 *Für jeden Graphen $G = (V, E)$ und jedes $A \subseteq V$ gilt:*

$$UN(A) = \{N(X) \cap \overline{A} : X \subseteq A\} = \{N(X) \cap \overline{A} : X \subseteq A \cap N(\overline{A})\}$$

Mit Blick auf diese Gleichheit ändert sich obige Fragestellung zu: Wie groß ist $|A \cap N(\overline{A})|$ mindestens und wie viele $X \subseteq A \cap N(\overline{A})$ haben paarweise verschiedene Nachbarschaften in $\overline{A}$? Mit diesen Fragen beschäftigt sich der nun folgende Abschnitt.

4.2 Dominierte und dominante Mengen

Unter Berücksichtigung der obigen Fragestellungen soll nun ein Kriterium eingeführt werden, das es für festes A ermöglicht, die Kardinalität von $UN(A)$ zu bestimmen. Dazu werden alle potenziellen $X \subseteq A$ hergenommen und danach klassifiziert, ob es eine größere Menge $X_0 \supsetneq X$ gibt, die die gleiche Nachbarschaft wie X in $\overline{A}$ hat.

Definition 4.2.1 *Es sei $G = (V, E)$ ein Graph und $A \subseteq V$. M_2 **dominiert** M_1 **über** A, kurz $M_1 \prec_A M_2$, falls $M_1 \subsetneq M_2 \subseteq A \cap N(\overline{A})$ ist und die Nachbarschaft von M_1 in $\overline{A}$ mit der von M_2 in $\overline{A}$ übereinstimmt. Formal heißt das also:*

$$M_1 \prec_A M_2 \quad :\Longleftrightarrow \quad M_1 \subsetneq M_2 \subseteq A \cap N(\overline{A}) \ \wedge \ N(M_1) \cap \overline{A} = N(M_2) \cap \overline{A}$$

*Die Knoten $v \in M_2 \backslash M_1$ heißen dominierte Knoten (über M_1). Eine Menge, die durch keine andere dominiert wird, heißt **dominant** über A.*

Damit sind insbesondere sowohl $\emptyset$ als auch A selbst immer über A dominant. Ist im Folgenden $A \subseteq V$ aus dem Zusammenhang klar, bezieht sich "dominiert" und "dominant" stets auf A. Sofort aus der Definition folgt hier:

Lemma 4.2.2 *Eine Menge $M \subseteq A \cap N(\overline{A})$ ist genau dann dominant, wenn für alle $v \in \left(A \cap N(\overline{A})\right) \backslash M$ die Bedingung $N(v) \cap \overline{A} \not\subseteq N(M) \cap \overline{A}$ gilt.*

Im Zuge dessen soll nun noch einmal kurz ein Blick auf das Beispiel aus Abbildung 3.1 bzw. 4.1 geworfen werden. Hier wurden zwei Schnitte $(A, \overline{A})$ und $(B, \overline{B})$ in einem (4×4)-Gittergraphen gezeigt, so dass zwar $|A \cap N(\overline{A})| = 4 = |B \cap N(\overline{B})|$ galt,

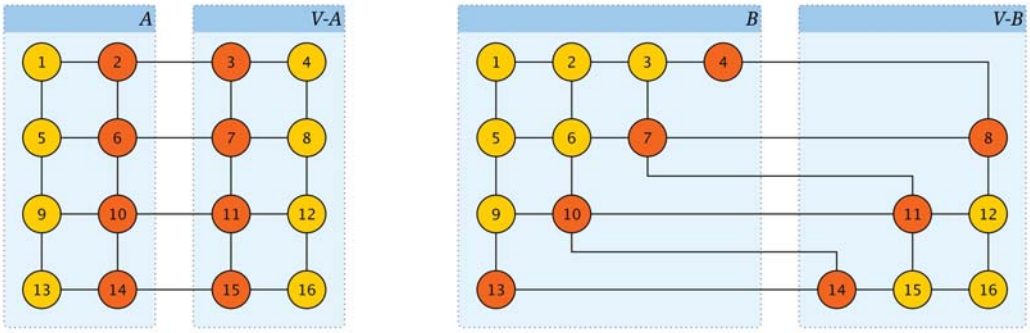

Abbildung 4.1: Zwei Schnitte $(A, \overline{A})$ und $(B, \overline{B})$ in einem (4×4)-Gittergraphen.

aber $UN(A)$ insgesamt aus 16 Mengen bestand, während $UN(B)$ nur sieben Mengen beinhaltete. Entsprechend hat $A \cap N(\overline{A})$ ausschließlich dominante Teilmengen, da keine zwei verschiedenen $M_1, M_2 \subseteq A \cap N(\overline{A})$ die gleiche Nachbarschaft in $\overline{A}$ haben.

Etwas anders stellt sich die Situation im Schnitt $(B, \overline{B})$ dar, denn hier existieren insgesamt nur sieben dominante Mengen. Dies ist insofern bemerkenswert, da auch $|UN(B)| = 7$ galt. Vor diesem Hintergrund ist die folgende Analyse zu sehen.

Lemma 4.2.3 *Sei $X \subseteq N(A) \cap \overline{A}$ fest vorgegeben. Dann gilt genau einer der beiden folgenden Fälle:*

1. $\{M \subseteq A \cap N(\overline{A}) : N(M) \cap \overline{A} = X\} = \emptyset$
2. $\{M \subseteq A \cap N(\overline{A}) : N(M) \cap \overline{A} = X\}$ *beinhaltet eine eindeutige dominante Menge*

Dabei gilt der erste Fall genau dann, wenn $X \notin UN(A)$.

Beweis
Es ist klar, dass sich die beiden Fälle ausschließen. Somit sei nun angenommen, dass $\{M \subseteq A \cap N(\overline{A}) : N(M) \cap \overline{A} = X\} \neq \emptyset$ gilt. Nach Konstruktion ist

$$M_\prec := M_{\prec A}(X) := \{v \in A \cap N(\overline{A}) : N(v) \cap \overline{A} \subseteq X\}$$

dominant über A und eindeutig durch X bestimmt. Da $N(M_\prec) \cap \overline{A} = X$ zu zeigen ist, sei angenommen, dass $N(M_\prec) \cap \overline{A} \neq X$ gilt. Nach Konstruktion erfüllt jedes $v \in M_\prec$, dass $N(v) \cap \overline{A} \subseteq X$ ist, aus $N(M_\prec) \cap \overline{A} \neq X$ folgt also $N(M_\prec) \cap \overline{A} \subsetneq X$.

Es ist $\{M \subseteq A \cap N(\overline{A}) : N(M) \cap \overline{A} = X\} \neq \emptyset$, also wähle ein beliebiges M mit $N(M) \cap \overline{A} = X$. Hier kann o.B.d.A. angenommen werden, dass $M_\prec \subseteq M$ gilt, denn sonst wähle $M := M \cup M_\prec$. Da aber $N(M_\prec) \cap \overline{A} \subsetneq X$ gilt, ist $M \backslash M_\prec \neq \emptyset$ und jedes $v \in M \backslash M_\prec$ erfüllt damit notwendigerweise:

$$\forall v \in M \backslash M_\prec \neq \emptyset : \ \exists w \in X \backslash N(M_\prec) : w \in N(v) \cap \overline{A} \quad \text{da } v \notin M_\prec$$
$$\forall v \in M \backslash M_\prec \neq \emptyset : \ N(v) \cap \overline{A} \subseteq X \quad \text{da } N(M) \cap \overline{A} = X$$

Die beiden Bedingungen widersprechen sich, also gilt $N(M_\prec) \cap \overline{A} = X$. Für den Zusatz ist nun noch zu zeigen, dass $\{M \subseteq A \cap N(\overline{A}) : N(M) \cap \overline{A} = X\}$ genau dann nicht-leer ist, wenn $X \in UN(A)$ gilt. Das folgt aber sofort aus der Definition von $UN(A)$. □

Aus Lemma 4.2.3 folgt also, dass es eine wohldefinierte Abbildung

$$M_{\prec A} : UN(A) \to \{M \subseteq A \cap N(\overline{A}) : M \text{ ist dominant über } A\}, \ X \mapsto M_{\prec A}(X)$$

gibt. Diese ist bijektiv, wie folgender Satz zeigt:

Satz 4.2.4 *Es sei $A \subseteq V(G)$ und $M_{\prec A}$ wie oben. Dann ist $M_{\prec A}$ bijektiv.*

Beweis
Für $X_1, X_2 \in UN(A)$ mit $X_1 \neq X_2$ seien $M_1 := M_{\prec A}(X_1)$ und $M_2 := M_{\prec A}(X_2)$. Dann gilt:

$$N(M_1) \cap \overline{A} = X_1 \neq X_2 = N(M_2) \cap \overline{A}$$

Also ist $M_1 \neq M_2$, d.h. $M_{\prec A}$ ist injektiv. Nun sei $M_0 \subseteq A \cap N(\overline{A})$ dominant sowie $X := N(M_0) \cap \overline{A}$. Dann ist $X \in UN(A)$, also enthält

$$\{M \subseteq A \cap N(\overline{A}) : N(M) \cap \overline{A} = X\}$$

nach Lemma 4.2.3 eine eindeutige dominante Menge $M_\prec = M_{\prec A}(X)$. Es ist aber auch $M_0 \in \{M \subseteq A \cap N(\overline{A}) : N(M) \cap \overline{A} = X\}$, also ergibt sich $M_0 = M_\prec$ und somit, dass $M_{\prec A}$ auch surjektiv ist und die Behauptung folgt. □

Wenn endliche Mengen in Bijektion zueinander stehen, haben sie bekanntlich gleich viele Elemente. Damit ergibt sich eine wichtige Folgerung.

Folgerung 4.2.5 *Es sei $A \subseteq V(G)$. Dann gilt:*

$$\begin{aligned} |UN(A)| &= |\{M \subseteq A \cap N(\overline{A}) : M \textit{ ist dominant über } A\}| \\ &= 2^{|A \cap N(A)|} - |\{M_1 \subseteq A \cap N(\overline{A}) \mid \exists M_2 \subseteq A \cap N(\overline{A}) : M_1 \prec_A M_2\}| \end{aligned}$$

Um also $|UN(A)|$ zu bestimmen, müssen jetzt "nur" noch die dominierten (bzw. dominanten) Mengen gezählt werden. Mit Lemma 4.2.3 ist hierzu ein nützliches Werkzeug gegeben. Die weitere Analyse beschäftigt sich jetzt noch mit der Frage, wann die Vereinigung zweier dominanter Mengen ebenfalls wieder dominant ist. Mit Lemma 4.2.2 folgt, dass $M_1 \cup M_2$ genau dann dominant ist, wenn für alle $v \in A \cap N(\overline{A})$ mit $v \notin M_1 \cup M_2$ gilt:

$$N(v) \cap \overline{A} \not\subseteq N(M_1 \cup M_2) \cap \overline{A}$$

Interessant ist es hier aber etwas genauer zu untersuchen was es bedeutet, wenn $M_1, M_2 \subseteq A \cap N(\overline{A})$ zwar beide über A dominant sind, $M_1 \cup M_2$ aber dominiert wird. Dann muss es einen Knoten $v \notin M_1 \cup M_2$ geben mit $N(v) \cap \overline{A} \subseteq N(M_1 \cup M_2) \cap \overline{A}$. Da aber M_1 und M_2 dominant sind, gilt sowohl $N(v) \cap \overline{A} \not\subseteq N(M_1) \cap \overline{A}$ als auch $N(v) \cap \overline{A} \not\subseteq N(M_2) \cap \overline{A}$, vgl. Lemma 4.2.2. Insgesamt heißt das, dass es $u \in N(M_1) \cap \overline{A}$ sowie $w \in N(M_2) \cap \overline{A}$ gibt mit $\{u, v\} \in E(G)$ und $\{v, w\} \in E(G)$, es gibt also einen 2-Pfad von $N(M_1)$ zu $N(M_2)$. Dies soll kurz festgehalten werden:

Folgerung 4.2.6 *Sind $M_1, M_2 \subseteq A \cap N(\overline{A})$ beide dominant und wird $M_1 \cup M_2$ dominiert, so gibt es*

1. *ein $v \in A \cap N(\overline{A})$ mit $v \notin M_1 \cup M_2$,*
2. *ein $u \in N(M_1) \cap \overline{A}$ mit $\{u, v\} \in E(G)$*
3. *und ein $w \in N(M_2) \cap \overline{A}$ mit $\{v, w\} \in E(G)$.*

Insbesondere gibt es einen Pfad $P_2 = (u, v, w)$ von $N(M_1) \cap \overline{A}$ zu $N(M_2) \cap \overline{A}$.

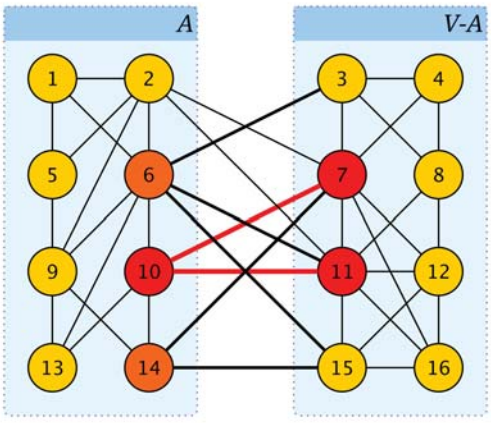

Abbildung 4.2: Eine Visualisierung der Situation aus Folgerung 4.2.6 mit markiertem Pfad P_2.

Zur Veranschaulichung der Situation aus Folgerung 4.2.6 ist ein Beispielgraph in Abbildung 4.2 gegeben. Hier sind sowohl $M_1 = \{6\}$ als auch $M_2 = \{14\}$ dominant über $A = \{1, 2, 5, 6, 9, 10, 13, 14\}$, aber $M_1 \cup M_2 = \{6, 14\}$ wird von $\{6, 10, 14\}$ dominiert, da die Nachbarschaften beider Mengen in $\overline{A}$ gleich sind. Hier ergibt sich der Pfad damit als $P_2 = (7, 10, 11)$. Diese Aussage wird später noch wichtig werden, zunächst aber eine weitere einfache Folgerung, die gut in Abbildung 4.3 verifiziert werden kann.

Folgerung 4.2.7 *Wieder sei $A \subseteq V(G)$ und $M_1, M_2 \subseteq A \cap N(\overline{A})$ dominant über A. Weiter sei $X_1 := N(M_1) \cap \overline{A}$ sowie $X_2 := N(M_2) \cap \overline{A}$. Gilt $N(X_1) \cap A \cap N(X_2) = \emptyset$, so ist $M_1 \cup M_2$ dominant.*

Wird A als variabel angenommen, ergeben sich damit bestimmte Knoten in $A \cap N(\overline{A})$ als besonders ungeeignet zur Minimierung von $|UN(A)|$.

Lemma 4.2.8 *Es sei $A \subseteq V(G)$ sowie $B \subseteq A \cap N(\overline{A})$ und $v \in B$. Gilt für die Menge $X = N(v) \cap \overline{A}$, dass $N(X) \cap A = \{v\}$ ist, so folgt:*

$$|\{M \subseteq B : M \text{ ist dominant über } A\}| = 2 \cdot |\{M \subseteq B - \{v\} : M \text{ ist dominant über } A\}|$$

Beweis
Wähle ein beliebiges dominantes $M \subseteq (B - \{v\}) \cap N(\overline{A})$. Dann ist M insbesondere in der ersten Menge enthalten. Weiter ist $\{v\}$ ebenfalls dominant, wie aus Lemma 4.2.2 mit der Voraussetzung an $X = N(v) \cap \overline{A}$ sofort folgt. Diese Voraussetzung impliziert ferner, dass $N(v) \cap \overline{A} \cap N(M) = \emptyset$ ist. Also induziert M eine weitere dominante Menge $\widetilde{M} := M \cup \{v\}$, wie Folgerung 4.2.7 liefert. Offensichtlich induzieren verschiedene M_1 und M_2 hier ebenfalls verschiedene $\widetilde{M_1}$ und $\widetilde{M_2}$, so dass die Behauptung folgt.□

Ein Beispiel zur Situation aus Lemma 4.2.8 ist in Abbildung 4.3 gegeben. Hier ist Knoten $v = 6$ der eindeutige Nachbar von $X = N(v) \cap \overline{A} = \{3, 15\}$ in A. Die über A dominanten Teilmengen von $A - \{v\}$ sind dann genau die beiden Mengen $\emptyset$ und $\{2, 10, 14\}$. Mit Folgerung 4.2.5 und Lemma 4.2.8 ergibt sich also $|UN(A)| = 2 \cdot 2 = 4$. Weitere Beispiele für Situationen, in denen Lemma 4.2.8 anwendbar ist, sind isolierte Knoten in A oder solche, deren zugehörige Schnittkanten ein Matching induzieren.

Zur Schranke aus Lemma 4.2.8 ist weiterhin noch zu sagen, dass sie außerdem den schlechtest möglichen Fall beschreibt, wie bereits auch schon in Abbildung 3.1 bzw. 4.1 zu sehen war. Dazu sei allgemein ein beliebiges $k \in \mathbb{N}$ sowie der Gittergraph G_k gegeben. Als Schnittmenge wähle dann ein $1 < \ell < k$ sowie

$$A := \{v_{i,j} : 1 \leq i \leq \ell,\ 1 \leq j \leq k\},$$

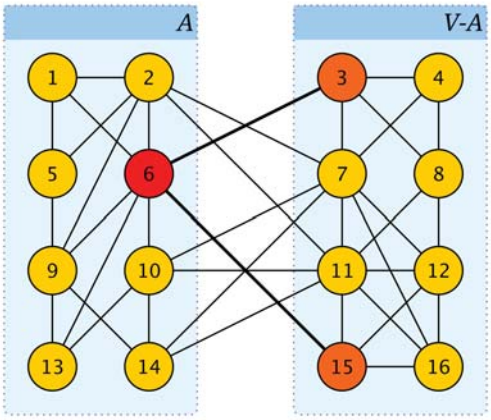

Abbildung 4.3: Eine Visualisierung der Situation aus Folgerung 4.2.7 und Lemma 4.2.8.

d.h. A besteht genau aus den ersten ℓ Zeilen des Gitters und die Schnittkanten induzieren ein perfektes Matching zwischen Zeile ℓ und $\ell+1$, mit jeweils Kanten zwischen $v_{\ell,j}$ und $v_{\ell+1,j}$. Dann gilt $A \cap N(\overline{A}) = k$ sowie $|UN(A)| = 2^k$, da keine zwei Mengen $M_1, M_2 \subseteq A \cap N(\overline{A})$, $M_1 \neq M_2$ die gleiche Nachbarschaft in $\overline{A}$ besitzen. Wird nun Lemma 4.2.8 k-Mal iteriert, ergibt sich ebenfalls $|UN(A)| = 2^k$.

Abschließend soll noch folgender wichtiger Satz angeführt werden. Dieser wird später ein paar Vereinfachungen ergeben.

Satz 4.2.9 (Kim [1982]) *Für jeden Graphen $G = (V, E)$ und jedes $A \subseteq V$ gilt* $|UN(A)| = |UN(\overline{A})|$.

4.3 Analyse der $(k \times k)$-Gittergraphen

Ziel ist es nun, festzustellen wie die Boolsche Weite eines $(k \times k)$-Gittergraphens ist. Dazu sei kurz an Definition 3.4.14 erinnert: Ein $(k \times k)$-Gitter ist der Graph $G_k = (V, E)$ mit

- $V = \{v_{i,j} : 1 \leq i, j \leq k\}$
- $E = E_1 \uplus E_2$ wobei
  - $E_1 = \{\{v_{i,j},\ v_{i,j+1}\} : 1 \leq i \leq k,\ 1 \leq j \leq k-1\}$
  - $E_2 = \{\{v_{i,j},\ v_{i+1,j}\} : 1 \leq i \leq k-1,\ 1 \leq j \leq k\}$

Da dies später noch relevant sein wird, sei folgender Zusatz erwähnt: Die Knoten $v \in V$ mit $d(v) = 4$ heißen **innere Knoten**, die übrigen werden **Randknoten** genannt. Besondere Randknoten sind die **Eckknoten**. Diese sind $v_{1,1}$, $v_{1,k}$, $v_{k,1}$ und $v_{k,k}$.

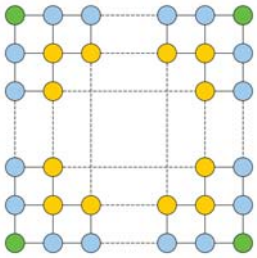

Abbildung 4.4: Gittergraph mit inneren, Rand- und Eckknoten

Im Folgenden soll zunächst gezeigt werden, welche kritischen Eigenschaften ein optimaler vollständiger Zerlegungsbaum für G_k auf jeden Fall erfüllt, um damit das Problem, die Boolsche Weite von G_k zu bestimmen, darauf zu reduzieren, die Boolsche Weite eines festen Zerlegungsbaumes zu berechnen. Dieser wird anschließend konstruiert, so dass sich die Frage nach der Boolschen Weite des Zerlegungsbaumes - und damit auch die nach der Boolschen Weite von G_k selbst - wiederum darauf reduziert, für ein festes A die Kardinalität von $UN(A)$ zu bestimmen. Letzteres Problem wird dann mit Hilfe der in Abschnitt 4.2 vorgestellten Methoden gelöst. Abschließend werden diese Ergebnisse noch auf $(k \times \ell)$-Gitter mit $k \leq \ell$ übertragen.

4.3.1 Reduktion: Boolsche Weite zur Kardinalität von *UN(A)*

Ziel dieses Abschnitts ist es, über nachweisbare Eigenschaften an einen optimalen vollständigen Zerlegungsbaum, das Problem die Boolsche Weite von G_k zu bestimmen auf die Berechnung der Kardinalität von $UN(A)$ zu reduzieren, wobei A eine entsprechend gewählte Menge ist. Ein erstes Ergebnis dazu formuliert folgender Satz.

Satz 4.3.1 *Es sei (T, δ) ein vollständiger Zerlegungsbaum für den Graphen G mit $|V(G)| = n$. Dann existiert ein $t \in T$ mit $|\delta(t)| \geq \lceil \frac{n}{3} \rceil$ und $|\overline{\delta(t)}| \geq \lceil \frac{n}{3} \rceil$.*

Beweis
Betrachte im i-ten Schritt jeweils einen Knoten $r = r_i \in T$ mit Söhnen $s_1 = s_{i,1}$ und $s_2 = s_{i,2}$, wobei o.B.d.A. immer $|\delta(s_{i,1})| \leq |\delta(s_{i,2})|$ gelte. Für r_1 gilt nach Definition $\delta(r_1) = V$. Dann gibt es zwei Fälle:

- $|\delta(s_{1,1})| \geq \lceil \frac{n}{3} \rceil$

  Dann folgt die Behauptung, da $|\delta(s_{1,1})| \leq |\delta(s_{1,2})|$.

- $1 \leq |\delta(s_{i,1})| < \lceil \frac{n}{3} \rceil$

  In $s_{i,1}$ werden also mindestens einer und höchstens $\lceil \frac{n}{3} \rceil - 1$ Knoten aus r_i herausgenommen, während die übrigen zwangsläufig zu $s_{i,2}$ gehören. Betrachte dann rekursiv $r_{i+1} := s_{i,2}$ mit $s_1 := s_{i+1,1}$ und $s_2 := s_{i+1,2}$. Dabei sind $s_{i+1,1}$ und $s_{i+1,2}$ die Söhne von r_i, die nach Konstruktion immer existieren.

Unterscheide dann im nächsten Schritt, ob $|\delta(s_{i,1})| + \sum_{j<i} |\delta(s_{j,1})| \leq \lceil \frac{n}{3} \rceil$ gilt oder nicht. Nach $\ell \leq \lceil \frac{n}{3} \rceil$ Iterationen tritt dann zwangsläufig folgende Situation ein:

(1) $\sum_{j=1}^{\ell} |\delta(s_{j,1})| \geq \lceil \frac{n}{3} \rceil$, da in jeder Iteration mindestens ein weiterer Knoten entfernt wird.

(2) $\sum_{j=1}^{\ell-1} |\delta(s_{j,1})| < \lceil \frac{n}{3} \rceil$, denn sonst betrachte $\ell := \ell - 1$.

Die Bedingungen (1) und (2) bedeuten also, dass ℓ minimal gewählt wird, so dass Bedingung (1) gilt. Weiter ist $\delta(r_\ell) = \delta(s_{1,\ell}) \uplus \delta(s_{2,\ell})$, also folgt:

$$|\delta(r_\ell)| = |\delta(s_{\ell,1})| + |\delta(s_{\ell,2})| = n - \sum_{j=1}^{\ell-1} |\delta(s_{j,1})| \overset{(2)}{>} n - \left\lceil \frac{n}{3} \right\rceil$$

Ferner ergibt sich mit $|\delta(s_{\ell,1})| \leq |\delta(s_{\ell,2})|$:

$$\begin{aligned} \Rightarrow n - \left\lceil \frac{n}{3} \right\rceil &< |\delta(s_{\ell,1})| + |\delta(s_{\ell,2})| \leq 2 \cdot |\delta(s_{\ell,2})| \\ \Rightarrow |\delta(s_{\ell,2})| &> \frac{1}{2} \cdot \left(n - \left\lceil \frac{n}{3} \right\rceil \right) \geq \frac{1}{2} \cdot \left(\left\lceil \frac{n}{3} \right\rceil + \left\lfloor \frac{n}{3} \right\rfloor + \left\lfloor \frac{n}{3} \right\rfloor - \left\lceil \frac{n}{3} \right\rceil \right) \\ &= \frac{1}{2} \cdot \left(\left\lfloor \frac{n}{3} \right\rfloor + \left\lfloor \frac{n}{3} \right\rfloor \right) = \left\lfloor \frac{n}{3} \right\rfloor \Longrightarrow |\delta(s_{\ell,2})| \geq \left\lceil \frac{n}{3} \right\rceil \end{aligned}$$

Da $\overline{\delta(s_{2,\ell})} = \biguplus_{j=1}^{\ell} \delta(s_{1,j})$ ist, folgt:

$$\left| \overline{\delta(s_{2,\ell})} \right| = \left| \biguplus_{j=1}^{\ell} \delta(s_{1,j}) \right| = \sum_{j=1}^{\ell} |\delta(s_{j,1})| \overset{(1)}{\geq} \left\lceil \frac{n}{3} \right\rceil$$

Die Behauptung gilt also für $s_{2,\ell}$. □

Mit Blick auf eine der ausgehenden Fragestellungen, wie groß $|A \cap N(\overline{A})|$ mindestens ist, soll nun gezeigt werden, dass jedes t aus Satz 4.3.1 unweigerlich eine Menge A mit $|A \cap N(\overline{A})| \geq k$ induziert. Für den Beweis dieser Aussage sei kurz angemerkt:

Bemerkung 4.3.2 *Für $1 \leq \ell \leq k$ sei mit $Z_\ell := \{v_{\ell,j} \in V(G_k) : 1 \leq j \leq k\}$ und $S_\ell := \{v_{i,\ell} \in V(G_k) : 1 \leq i \leq k\}$ jeweils die komplette ℓ-te Zeile bzw. die komplette ℓ-te Spalte des Gitters G_k bezeichnet. Dann gibt es für $A \subset V(G_k)$ mit $|A| \leq |\overline{A}|$ und $|A \cap N(\overline{A})| \leq k-1$ kein $1 \leq \ell \leq k$ mit $Z_\ell \subseteq A$ oder $S_\ell \subseteq A$.*

Damit kann nun der folgende Satz gezeigt werden.

Satz 4.3.3 *Ist $A \subset V(G_k)$ mit $k \leq |A| \leq |\overline{A}|$ und $|A \cap N(\overline{A})| \leq k-1$, so folgt $|A| \leq \frac{k\cdot(k-1)}{2}$. Die Schranke ist scharf.*

Beweis
Es gelte $|A \cap N(\overline{A})| = k-1$, denn sonst wähle $A' \supset A$ mit $|A' \cap N(\overline{A'})| = k-1$. Die Gitterstruktur von G_k stellt sicher, dass ein solches A' existiert (verschiebe entsprechende Knoten von $\overline{A}$ zu A). Gilt die Behauptung dann für A', so trifft sie auch für A zu.

Allgemein gibt es nun zwei Fälle:

- $G_k[A]$ ist zusammenhängend. Betrachte die beiden Diagonalen in G_k:

$$\text{Diag}_1 := \{v_{i,i} \in V(G_k) : 1 \leq i \leq k\}$$
$$\text{Diag}_2 := \{v_{j,k-j+1} \in V(G_k) : 1 \leq j \leq k\}$$

  Dann gibt es zwei Möglichkeiten für A. Entweder schneidet sich A mit einer Diagonalen nicht, d.h. es gibt $i \in \{1,2\}$ mit $A \cap \text{Diag}_i = \emptyset$, oder A schneidet sich mit beiden Diagonalen. Gilt Ersteres, so folgt die Behauptung, denn $V(G_k) - \text{Diag}_i$ besitzt genau zwei Komponenten mit je $\frac{k\cdot(k-1)}{2}$ Knoten und A ist zusammenhängend, also in genau einer der beiden enthalten. Schneidet sich A mit beiden Diagonalen, so unterscheide weitere zwei Fälle:

  - A beinhaltet maximal einen Randknoten und besteht sonst nur aus inneren Knoten, d.h. Knoten aus beiden Diagonlen werden durch $A \cap N(\overline{A})$ vom Rand des Gitters getrennt. Dann ist klar, dass mit $|A \cap N(\overline{A})| = k-1$ so nie mehr als $\frac{k\cdot(k-1)}{2}$ viele Knoten umrandet werden können.
  - Es gibt ein $v_{i_0,i_0} \in \text{Diag}_1$, ein $v_{j_0,k-j_0+1} \in \text{Diag}_2$ und (mindestens) zwei Randknoten in $A \cap N(\overline{A})$.

    Dabei kann i_0 so gewählt werden, dass $v_{i,i} \notin A$ für alle $i < i_0$. Analog kann j_0 so gewählt werden, dass $v_{j_0,k-j_0+1} \notin A$ für alle $j < j_0$. Aufgrund der Symmetrie des Graphen kann weiter o.B.d.A. angenommen werden, dass sowohl $i_0 \leq \left\lceil \frac{k}{2} \right\rceil$ als auch $j_0 \leq \left\lceil \frac{k}{2} \right\rceil$ erfüllt ist.

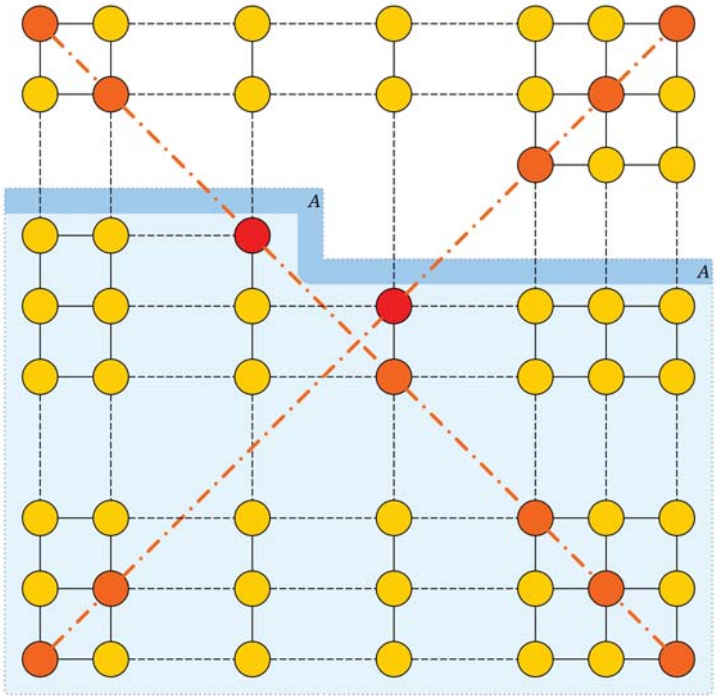

Abbildung 4.5: Gittergraph mit Knotenmenge A, den beiden Diagonalen Diag_1 und Diag_2 sowie den Knoten $v_{i_0,i_0} \in \mathsf{Diag}_1$ und $v_{j_0,k-j_0+1} \in \mathsf{Diag}_2$.

Ist $i_0 \geq 2$, ergeben sich so mindestens $i_0 - 1$ viele Knoten in $A \cap N(\overline{A})$, die notwendig sind, um v_{i_0,i_0} von $v_{1,1}$ zu trennen; für $j_0 \geq 2$ ergeben sich wieder analog mindestens $j_0 - 1$ weitere Knoten, um $v_{j_0,k-j_0+1}$ von $v_{1,k}$ zu trennen. So ergeben sich also zusammen $(i_0-1)+(j_0-1)$ viele Knoten in $A \cap N(\overline{A})$. Die Menge dieser Knoten sei $\tilde{A}$.

Werden i_0 und j_0 mitgezählt, so gibt es genau $k - [(i_0 - 1) + (j_0 - 1)]$ Spalten zwischen i_0 und j_0, von denen jede nicht-leeren Schnitt mit A hat (denn A hat je einen Knoten in i_0 und j_0 und ist zusammenhängend) und entweder mindestens einen weiteren Knoten aus $A \cap N(\overline{A})$ enthält, oder es gibt eine Spalte $S_\ell = \{v_{i,\ell} : 1 \leq i \leq k\}$ für $i_0 \leq \ell \leq j_0$, die ganz in A liegt, also $S_\ell \subset A$. Letzteres scheidet nach den Einsichten von Bemerkung 4.3.2 aus, jede Spalte zwischen i_0 und j_0 enthält also mindestens einen Knoten aus $A \cap N(\overline{A})$. Es folgt

$$|A \cap N(\overline{A})| \geq |\tilde{A}| + k - [(i_0 - 1) + (j_0 - 1)] \geq k,$$

was ein Widerspruch ist. Also schneidet sich A mit mindestens einer Diagonalen trivial und die Behauptung folgt.

- $G_k[A]$ ist nicht zusammenhängend:

  Betrachte die Zusammenhangskomponenten $D_1, \ldots, D_d$ sowie $D_{d+1}, \ldots, D_{d+r}$ von $G_k[A]$, so dass $|V(D_i) \cap N(\overline{A})| = k_i > 1$ für $i = 1, \ldots, d$ und weiter $|V(D_i)| = k_i = 1$ für $i = d+1, \ldots, d+r$ gilt. Dann ist $\sum_{i=1}^{d+r} k_i = k-1$ und für jedes $i = 1, \ldots, d$ gilt $|V(D_i)| \leq \frac{k_i \cdot (k_i - 1)}{2}$ (s.o.). Insgesamt folgt daher:

$$
\begin{aligned}
|A| = \sum_{i=1}^{d+r} |V(D_i)| &= \sum_{i=1}^{d} |V(D_i)| + \sum_{i=d+1}^{d+r} k_i \leq \sum_{i=1}^{d} \frac{k_i \cdot (k_i - 1)}{2} + \sum_{i=d+1}^{d+r} k_i \\
&= \frac{1}{2} \left(\sum_{i=1}^{d} k_i^2 - \sum_{i=1}^{d} k_i + \sum_{i=d+1}^{d+r} 2k_i \right) &&= \frac{1}{2} \left(\sum_{i=1}^{d} k_i^2 + \sum_{i=d+1}^{d+r} 3k_i - \sum_{i=1}^{d+r} k_i \right) \\
&\leq \frac{1}{2} \left(\sum_{i=1}^{d+r} k_i \sum_{j=1}^{d} k_j + \sum_{i=d+1}^{d+r} 3k_i - \sum_{i=1}^{d+r} k_i \right) &&\leq \frac{1}{2} \left(\sum_{i=1}^{d+r} k_i \sum_{j=1}^{d+r} k_j - \sum_{i=1}^{d+r} k_i \right) \\
&= \frac{1}{2} \left(\left(\sum_{i=1}^{d+r} k_i \right)^2 - \sum_{i=1}^{d+r} k_i \right) &&= \frac{1}{2} \left((k-1)^2 - (k-1) \right) \\
&= \frac{(k-1)}{2} \cdot (k-1-1) &&\leq \frac{k \cdot (k-1)}{2}
\end{aligned}
$$

Für den Zusatz betrachte den Diagonalschnitt $DS := \{v_{i,j} \in V(G_k) : 1 \leq i < j \leq k\}$ mit $|DS| = \frac{k \cdot (k-1)}{2}$ und $|DS \cap N(\overline{DS})| = k-1$. Dann erfüllt DS alle Voraussetzungen und die gezeigte Schranke mit Gleichheit. □

Ist es nun die Aufgabe, $|UN(A)|$ zu minimieren, so ist es zunächst sinnvoll den Zerlegungsbaum so zu konstruieren, dass das maximierende A möglichst wenige Teilmengen hat, deren Nachbarschaft sich mit $\overline{A}$ nicht-trivial schneidet. Es gibt potenziell $2^{|A \cap N(\overline{A})|}$ solche Mengen in A, es liegt also nahe die Menge $A \cap N(\overline{A})$ so klein wie möglich zu halten.

Gezeigt ist mit den letzten beiden Sätzen zusammen, dass diese Schranke nicht unter k zu drücken ist (unter Verwendung von $|UN(A)| = |UN(\overline{A})|$, siehe Satz 4.2.9), denn jedes $t \in T$ mit $|\delta(t)| \geq \left\lceil \frac{k^2}{3} \right\rceil$ und $|\overline{\delta(t)}| \geq \left\lceil \frac{k^2}{3} \right\rceil$ erfüllt dann, dass $\max\{|\delta(t)|, |\overline{\delta(t)}|\} > \frac{k \cdot (k-1)}{2}$. Also folgt:

Folgerung 4.3.4 *Jedes $t \in T$ mit der Eigenschaft aus Satz 4.3.1 erfüllt:*

$$\max \left\{ |\delta(t) \cap N(\overline{\delta(t)})|, \ |\overline{\delta(t)} \cap N(\delta(t))| \right\} \geq k$$

Zurückblickend auf die ausgehenden Fragestellungen ist also festzuhalten, dass jeder vollständige Zerlegungsbaum eine Menge A mit $|A \cap N(\overline{A})| \geq k$ induziert. Ferner ergibt mit Folgerung 4.2.6 hier eine interessante Aussage.

Bemerkung 4.3.5 *Jede Strategie, die (global) möglichst wenig dominante Mengen hat, hat so wenig wie möglich Kanten in* $G_k\left[A \cap N(\overline{A})\right]$. *Denn jede Kante* $\{u, v\}$ *mit* $\{u, v\} \in E\left(G_k\left[A \cap N(\overline{A})\right]\right)$ *liefert, dass kein 2-Pfad von* $N(u) \cap \overline{A}$ *zu* $N(v) \cap \overline{A}$ *über* $A \cap N(\overline{A})$ *existieren kann.*

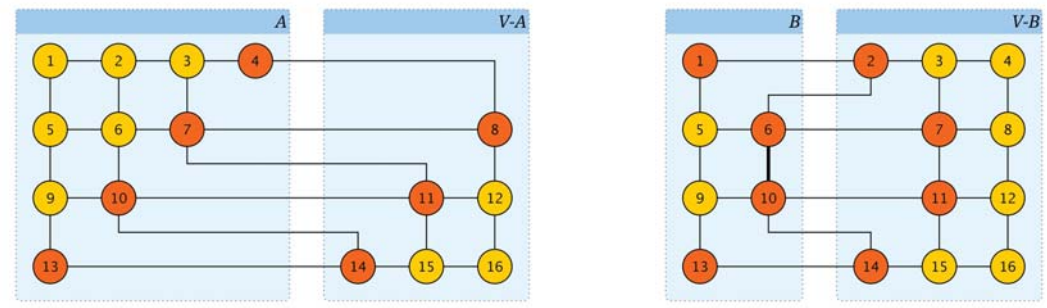

Abbildung 4.6: Eine Visualisierung der Aussage aus Bemerkung 4.3.5.

Um die Aussage von Bemerkung 4.3.5 etwas besser darstellen zu können, sei nochmals auf den Schnitt aus Abbildung 4.1 verwiesen. Den dortigen Anmerkungen ist zu entnehmen, dass für den Schnitt $(A, \overline{A})$ in Abbildung 4.6 $|UN(A)| = 7$ gilt. Nebenstehend ist eine Möglichkeit abgebildet, so dass die Menge $B \cap N(\overline{B})$ eine induzierte Kante hat und $|UN(B)|$ möglichst klein ist.

Eine genauere Analyse zeigt hier, dass die über B dominanten Mengen gerade $\emptyset$, $\{1\}$, $\{13\}$, $\{1, 6\}$, $\{10, 13\}$, $\{1, 6, 13\}$, $\{1, 10, 13\}$ und $\{1, 6, 10, 13\}$ sind. Mit Folgerung 4.2.5 ergibt sich also $|UN(B)| = 8 > 7 = |UN(A)|$.

Daher soll nun noch ein Schnitt $(A, \overline{A})$ konstruiert werden, der allen bisher gezeigten Eigenschaften genügt, auf den Lemma 4.2.8 oder Bemerkung 4.3.5 aber nicht anwendbar ist. Betrachte dazu den Schnitt $(D, \overline{D})$ in G_k mit:

$$D = \{v_{i,j} \in V(G_k) : 1 \leq i, j \leq k,\ i + j \leq k + 1\}$$
$$\overline{D} = \{v_{i,j} \in V(G_k) : 1 \leq i, j \leq k,\ i + j > k + 1\}$$

Die Menge $D \cap N(\overline{D})$ besteht dann aus der Diagonalen

$$D \cap N(\overline{D}) = \{v_{i,k-i+1} \in V(G_k) : 1 \leq i \leq k\},$$

an der G_k geteilt wird. Dann erfüllt der Schnitt alle kritischen Eigenschaften, d.h. es gilt

- $|D| \geq \left\lceil \frac{k^2}{3} \right\rceil$, $|\overline{D}| \geq \left\lceil \frac{k^2}{3} \right\rceil$
- $|D \cap N(\overline{D})| \geq k$ (sogar $|D \cap N(\overline{D})| = k$)
- $E\left(G\left[D \cap N(\overline{D})\right]\right) = \emptyset$

und es gibt keinen Knoten auf den Lemma 4.2.8 anwendbar ist.

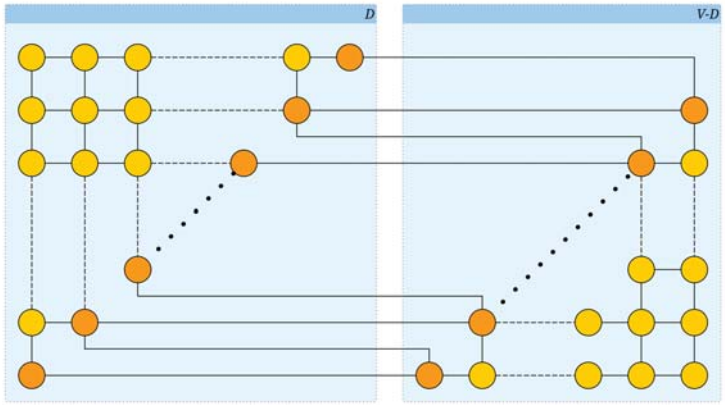

Abbildung 4.7: Der Gittergraph G_k mit Diagonalschnitt $(D, \overline{D})$.

Eine Veranschaulichung zum Schnitt $(D, \overline{D})$ im Gittergraphen G_k ist mit Abbildung 4.7 gegeben. Damit ist die Vorarbeit für einen wichtigen Satz getan:

Hauptsatz 4.3.6 *Es sei (T, δ) ein vollständiger Zerlegungsbaum, der boolw(G_k) für $k \geq 3$ minimiert. Dann gilt:*

$$\max_{t \in T} \max\{\ |\delta(t) \cap N(\overline{\delta(t)})|, |\overline{\delta(t)} \cap N(\delta(t))|\ \} = k$$

Das maximierende $t_{\max} \in T$ erfüllt außerdem, dass o.B.d.A. $\delta(t_{\max})$ einen Diagonalschnitt mit

$$\delta(t_{\max}) = \{v_{i,j} \in V(G_k) : 1 \leq i, j \leq k,\ i + j \leq k + 1\}$$
$$\overline{\delta(t_{\max})} = \{v_{i,j} \in V(G_k) : 1 \leq i, j \leq k,\ i + j > k + 1\}$$

definiert.

Beweis
Die Behauptung wird per vollständiger Induktion nach k gezeigt. Für $k = 3$ ergibt sich die Behauptung durch direktes Nachprüfen der möglichen Schnitte.

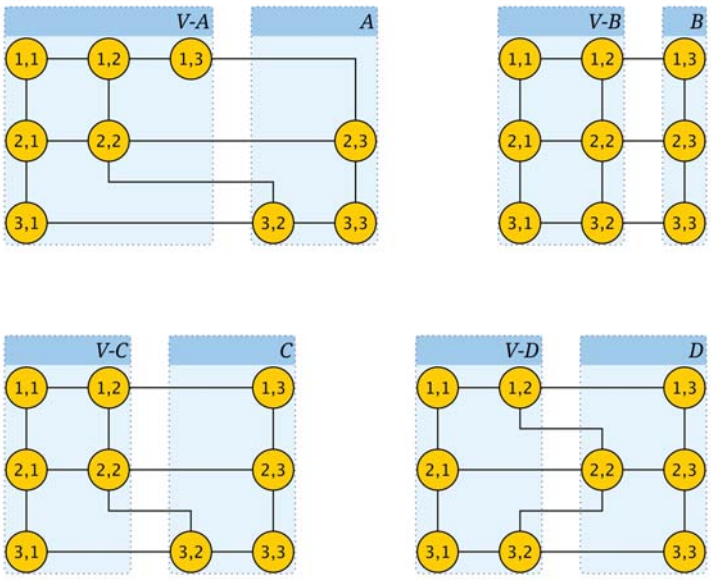

Abbildung 4.8: Vier Schnitte im Gittergraphen G_3.

Einige mögliche Ansätze sind in Abbildung 4.8 gegeben. Werden die jeweils dominanten Mengen bestimmt ergibt sich, dass A mit $|UN(A)| = 4$ minimal ist im Vergleich zu $|UN(B)| = 8$, $|UN(C)| = 6$ und $|UN(D)| = 5$. Mit den Lösungsmöglichkeiten aus Kapitel 5 folgt, dass es tatsächlich auch keine bessere Möglichkeit gibt (bis auf mögliche Symmetrien und Isomorphien, u.a. wegen Satz 4.2.9). Die Behauptung gilt also für $k = 3$.

Daher sei die Behauptung nun für alle $\ell \leq k$ erfüllt. Betrachte dann einen optimalen vollständigen Zerlegungsbaum (T, δ) für den Gittergraphen G_{k+1}. Nach Satz 4.3.1 und Folgerung 4.3.4 existiert ein $t \in T$, so dass

$$\max\left\{|\delta(t) \cap N(\overline{\delta(t)})|,\ |\overline{\delta(t)} \cap N(\delta(t))|\right\} \geq k+1$$

ist. Analysiere nun $UN(A)$ mit $A = \delta(t)$ (bzw. $A = \overline{\delta(t)}$): Folgerung 4.2.5 impliziert

$$|UN(A)| = |\{M \subseteq A \cap N(\overline{A}) : M \text{ ist dominant über } A\}|,$$

A erfüllt also $|A \cap N(\overline{A})| \geq k+1$ und A hat minimal viele dominante Teilmengen. Betrachte daher zwei dominante Mengen $M_1, M_2 \subseteq A \cap N(\overline{A})$. Nach Folgerung 4.2.7 ist die Vereinigung zweier dominanter Mengen $M_1 \cup M_2$ aber wieder dominant, wenn kein 2-Pfad zwischen ihnen in G über $\overline{A}$ existiert. Die Existenz solcher 2-Pfade zwischen dominanten Mengen kann aber nur lokal im Graphen gelten. Denn sobald beispielsweise zwei komplette Zeilen und/oder Spalten zwischen M_1 und M_2 liegen,

kann kein solcher Pfad existieren. Entsprechend muss A, um insgesamt minimal viele dominante Teilmengen zu haben, auch lokal optimal sein.

Wird nun

$$W_1 := \{v_{i,j} \in V(G_k) : i \neq k+1 \neq j\}$$
$$W_2 := \{v_{i,j} \in V(G_k) : i \neq 1 \neq j\}$$
$$W_3 := \{v_{i,j} \in V(G_k) : i \neq 1,\ j \neq k+1\}$$
$$W_4 := \{v_{i,j} \in V(G_k) : i \neq k+1,\ j \neq 1\}$$

definiert, so induziert (T, δ) mit Algorithmus 3.1 jeweils einen vollständigen Zerlegungsbaum (T_d, δ_d) für $G[W_d] \cong G_k$, $d \in \{1, 2, 3, 4\}$.

Mit der Induktionsvoraussetzung hat $A \cap W_d =: A_d$ in mindestens einem dieser Teilgraphen (bis auf Isomorphien und Symmetrien) die behauptete Struktur. Eine einfache Fallunterscheidung zu den Möglichkeiten in G_{k+1} einen weiteren Knoten zu A_d hinzuzufügen, liefert, dass die behauptete (wieder bis auf Symmetrien und Isomorphien) die einzige ist, die $|UN(\delta(t))|$ minimiert. Wird A dann wie im folgenden Abschnitt 4.3.2 weiter zerlegt, dann maximiert t bzw. A den Wert $|UN(\delta(t))|$ über T. Die Behauptung gilt also auch für $k+1$, was die Induktion beendet. □

Mit dem Hauptsatz 4.3.6 ist also das Problem, die Boolsche Weite von G_k zu bestimmen, darauf reduziert die Kardinalität von $UN(\delta(t_{\max}))$ zu berechnen. Dem wird im übernächsten Abschnitt nachgegangen, jetzt folgt erst noch ein kurzer Teil darüber, wie damit ein optimaler vollständiger Zerlegungsbaum für G_k zu konstruieren ist.

4.3.2 Konstruktion des optimalen vollständigen Zerlegungsbaumes

In diesem Abschnitt soll kurz noch darauf eingegangen, wie ein optimaler vollständiger Zerlegungsbaum, der alle Eigenschaften aus dem letzten Abschnitt genügt, konstruiert werden kann. Dazu sei k in diesem Abschnitt beliebig, aber fest gewählt. Hier sei vorab nochmals an die Aussagen aus Kapitel 3 erinnert: Offensichtlich ist für $\ell < k$ das Gitter G_ℓ ein induzierter Teilgraph von G_k, also liefert Satz 3.4.17 bzw. Algorithmus 3.1, dass $boolw(G_\ell) \leq boolw(G_k)$ ist.

Die verwendete Technik basiert, wie auch die letzten Aussagen aus dem letzten Abschnitt, auf Diagonalschnitten. Zur besseren Lesbarkeit wird hier nicht zwischen Knoten des Zerlegungsbaumes und ihren Labels unterschieden. Soll heißen: Ein Knoten $t \in T$ ist bereits eine Menge von Knoten, also $t \subseteq V(G_k)$. Da die Knotenmenge eines Zerlegungsbaumes nicht weiter spezifiziert ist, kann diese Einschränkung

getroffen werden, was den Vorteil hat, das δ nicht immer mit angegeben werden muss, da $\delta(t) = t$ nach Konstruktion stets erfüllt ist. Begonnen wird daher mit $T := (\{V(G_k)\}, \emptyset)$.

Als zweiter Schritt wird $V(G_k)$ gemäß dem Diagonalschnitt aus Hauptsatz 4.3.6 geteilt, d.h. es werden die Mengen

$$S_1 := \{v_{i,j} \in V(G_k) : 1 \leq i, j \leq k,\ i + j \leq k + 1\}$$
$$S_2 := \{v_{i,j} \in V(G_k) : 1 \leq i, j \leq k,\ i + j > k + 1\}$$

definiert und $\text{sons}(V(G_k)) := \{S_1, S_2\}$ gesetzt. Anschließend wird rekursiv auf S_1 und S_2 gearbeitet, es liegt im Folgenden also immer die Situation vor, dass eine Menge $A \subsetneq V(G_k)$ gegeben ist, die nicht-trivial partitioniert werden muss. Betrachte dann die Diagonalen $D_r := \{v_{i,j} \in A : i + j = r\} \subseteq A$. Offensichtlich gibt es nach Konstruktion eine eindeutige Diagonale D_{r_0} in A mit $|D_{r_0}| > |D_r|$ für alle $r \neq r_0$.

Ist $D_{r_0} \subsetneq A$, so setze $\text{sons}(A) := \{D_{r_0}, A \backslash D_{r_0}\}$ und arbeite erneut rekursiv auf beiden Söhnen. Gilt $D_{r_0} = A$, so sei v_0 der Knoten mit kleinstem ersten Index in D_{r_0}. Definiere dann $\text{sons}(A) := \{\{v_0\}, A - \{v_0\}\}$ und fahre rekursiv auf $A - \{v_0\}$ fort, falls $|A - \{v_0\}| > 1$ gilt.

Der Gittergraph wird also zunächst in zwei Dreiecke und dann sukzessive in Diagonalen zerlegt. Insbesondere ist die Menge $A \cap N(\overline{A})$ für alle $A \in T$ stets eine Diagonale (bzw. eine Teildiagonale) oder $|A| = 1$. Damit ist der so konstruierte vollständige Zerlegungsbaum zu G_k nach Hauptsatz 4.3.6 optimal.

Zum besseren Verständnis wird das erklärte Vorgehen nun noch etwas formaler als Pseudocode angegeben.

Listing 4.1: Algorithmus `Gk_fdt`

```
1  Gk_fdt(Gk) {
2     /* es wird mit Gk_fdt_rec ein optimaler vollständiger Zerlegungsbaum für Gk berechnet */
3
4     /* initialisiere wie oben */
5     S1 := { vi,j ∈ V(Gk) : 1 ≤ i, j ≤ k, i+j ≤ k+1 };
6     S2 := { vi,j ∈ V(Gk) : 1 ≤ i, j ≤ k, i+j > k+1 };
7     sons(V(Gk)) := { S1, S2 };
8
9     /* benutze Rekursion mit Gk_fdt_rec */
10    Gk_fdt_rec(S1);
11    Gk_fdt_rec(S2);
12 }
```

Listing 4.2: Algorithmus `Gk_fdt_rec`

```
1  Gk_fdt_rec(A) {
2     /* es wird A nach obigem Verfahren rekursiv zerlegt */
3
4     /* konstruiere die Diagonalen */
5     for 1 ≤ r ≤ k do
6        D_r := { v_{i,j} ∈ A : i+j ≤ r };
7
8     r_0 := arg max{ |D_r| : 1 ≤ r ≤ k };
9
10    /* jetzt ist die größte Diagonale D_{r_0} bekannt */
11
12    /* unterscheide, ob die A=D_{r_0} ist oder nicht */
13    if (A == D_{r_0})
14       v := A[1];
15       sons(A) := { {v}, A-{v} };
16
17       if (|A - {v}| > 1)
18          Gk_fdt_rec(A - {v});
19
20    else
21       sons(A) := { D_{r_0}, A\D_{r_0} };
22       Gk_fdt_rec(D_{r_0});
23       Gk_fdt_rec(A\D_{r_0});
24 }
```

Werden die Knoten eines (3×3)-Gittergraphen gemäß $\iota(v_{i,j}) := 3 \cdot (i-1) + j$ umbenannt (d.h. die Knoten werden von 1 bis 9 durchnummeriert), so ergibt sich mit dem in Abbildung 4.9 dargestellten vollständigen Zerlegungsbaum für G_3 eine Veranschaulichung des Algorithmus.

4.3.3 Genaue Bestimmung der Boolschen Weite

Im nun folgenden Teil wird der Frage nachgegangen, wie sich mit den bisherigen Ergebnissen die eigentliche Boolsche Weite der Gittergraphen G_k bestimmen lässt. Nach den Resultaten von Abschnitt 4.3 ist dazu nur noch die Anzahl der Elemente von $UN(\delta(t_{\max}))$ zu bestimmen, wobei $\delta(t_{\max})$ wie im Hauptsatz 4.3.6 ist, und diese Anzahl anschließend zu logarithmieren.

Das eigentliche Problem liegt also darin, $|UN(\delta(t_{\max}))|$ in Abhängigkeit der Größe des Graphen - also von k - zu bestimmen. Dies wird auf ein Abzählproblem zurückgeführt, das anschließend gelöst wird, indem eine Rekursionsgleichung angegeben wird. Abschließend erfolgt für diese noch eine asymptotische Abschätzung.

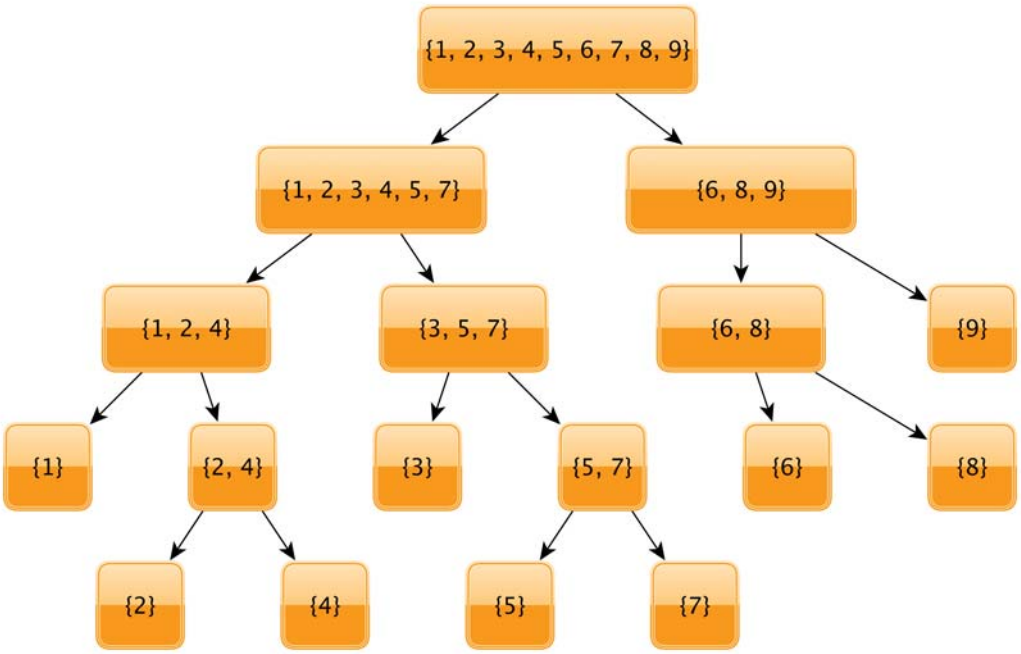

Abbildung 4.9: Ein optimaler vollständiger Zerlegungsbaum zu einem (3×3)-Gittergraphen, dessen Knoten von 1 bis 9 nummeriert sind. Konstruiert wurde er mit den Algorithmen 4.1 und 4.2.

Das zu lösende Problem besteht nun darin, $|UN(D)|$ für ein geeignetes $D = \delta(t_{\max})$ zu bestimmen. Dabei bildet D das obere Dreieck des $(k \times k)$-Gitters, also es ist

$$D = \{v_{i,j} \in V(G_k) : i + j \leq k + 1\}.$$

Weiter sei hier

$$A := \{v_{i,j} \in V(G_k) : i + j \geq k + 2\} = \overline{D}$$

Mit Satz 4.2.9 folgt, dass $|UN(D)| = |UN(\overline{D})| = |UN(A)|$ ist. Wie sich später zeigt, ist dies einfacher, als direkt mit $UN(\delta(t_{\max})) = UN(D)$ zu arbeiten. Unter Verwendung von Folgerung 4.1.2 lässt sich dies vereinfachen zu:

$$UN(A) = \{N(X) \cap \overline{A} : X \subseteq A \cap N(\overline{A})\}$$

Die Menge $A \cap N(\overline{A})$ soll kurz genauer untersucht werden. Hier gilt offensichtlich:

$$|UN(A)| = |UN(A \cap N(\overline{A}))| = |UN(\{v_{i,j} \in V(G_k) : i + j = k + 2\})|$$

Dieses Ergebnis soll kurz festgehalten werden.

Bemerkung 4.3.7 *Mit* $A_\cap := \{v_{i,j} \in V(G_k) : i + j = k + 2\}$ *und* A *wie oben gilt* $UN(A) = UN(A_\cap)$.

Die Menge $A_\cap$ ist also entscheidend für die Bestimmung von $UN(A)$. Interessant sind nun die Teilmengen von $A_\cap$ - denn diese stehen in Bijektion zu den Binärfolgen der Länge $k - 1$:

Lemma 4.3.8 *Es gibt eine Bijektion* $\varphi : Pot(A_\cap) \to \{0, 1\}^{k-1}$.

Beweis
Nach Definition ist

$$\begin{aligned} A_\cap &= \{v_{i,j} \in V(G_k) : i + j = k + 2\} \\ &= \{v_{i,j} \in V(G_k) : j = k - i + 2\} \\ &= \{v_{i,k-i+2} \in V(G_k) : 1 \leq i \leq k - 1\} \end{aligned}$$

sowie $Pot(A_\cap) = \{X : X \subseteq A_\cap\}$. Es sei daher $X \subseteq A$. Definiere $a = (a_i)_{1 \leq i \leq k-1}$ wie folgt:

$$a_i := \begin{cases} 1 & \text{wenn } v_{i,k-i+2} \in X \\ 0 & \text{wenn } v_{i,k-i+2} \notin X \end{cases}$$

Dann ist a eindeutig durch X und X eindeutig durch a bestimmt. Das ist die Behauptung. □

Der Nutzen dieser Bijektion erscheint zwar noch etwas unklar, dennoch wird sie noch sehr nützlich sein. Doch vorab eine kurze Definition.

Definition 4.3.9 *Für* $k \in \mathbb{N}$ *sei* μ_k *die Anzahl der Binärfolgen mit Länge* k*, die* 101 *nicht als Teilfolge enthalten:*

$$\mu_k := |\{a \in \{0, 1\}^k \mid \nexists 1 \leq i \leq n - 2 : a_i = 1 \ \wedge \ a_{i+1} = 0 \ \wedge \ a_{i+2} = 1\}|$$

Mit Hilfe dieser Definition und des folgenden Lemmas ist es nun möglich, die Berechnung von $|UN(A_\cap)| = |UN(A)|$ auf eine andere Ebene zu abstrahieren.

Lemma 4.3.10 *Mit* $A = \{v_{i,j} \in V(G_k) : i + j \geq k + 2\}$ *gilt* $|UN(A)| = \mu_{k-1}$.

Beweis
Es sei wie bisher $A_\cap := \{v_{i,j} \in V(G_k) : i+j = k+2\} = A \cap N(\overline{A})$. Unter Verwendung von Satz 4.2.9 sowie Folgerung 4.2.5 ist dann:

$$|UN(A)| = |UN(A_\cap)| = |\{M \subseteq A_\cap : M \text{ ist dominant über } A_\cap\}|$$

Nach Definition 4.2.1 ist $M \subseteq A_\cap$ genau dann dominant, wenn kein $M_0 \supsetneq M$ mit $M_0 \subseteq A_\cap$ existiert, so dass $N(M_0) \cap \overline{A_\cap} = N(M) \cap \overline{A_\cap}$ gilt. Definiere für alle $1 \leq i \leq k-1$ hier $v_i := v_{i,k-i+2} \in V(G_k)$. Dann gilt:

$$
\begin{aligned}
& M \subseteq A_\cap \text{ ist dominant} \\
\iff & \nexists M_0 \subseteq A_\cap && : M \subsetneq M_0 \ \wedge \ N(M_0) \cap \overline{A_\cap} = N(M) \cap \overline{A_\cap} \\
\iff & \forall v \in A_\cap \backslash M && : N(v) \cap \overline{A_\cap} \not\subseteq N(M) \cap \overline{A_\cap} \\
\iff & \forall v_i \in A_\cap \backslash M && : 1 < i < k-1 \ \Rightarrow \ (v_{i-1} \notin A_\cap \ \vee \ v_{i+1} \notin A_\cap) \\
\overset{4.3.8}{\iff} & \forall 1 < i < k-1 && : \varphi(A_\cap)_i = 0 \Rightarrow (\varphi(A_\cap)_{i-1} \neq 1 \ \vee \ \varphi(A_\cap)_{i+1} \neq 1) \\
\iff & \forall 1 < i < k-1 && : \varphi(A_\cap)_i = 0 \Rightarrow (\varphi(A_\cap)_{i-1} = 0 \ \vee \ \varphi(A_\cap)_{i+1} = 0) \\
\iff & \nexists 1 < i < k-1 && : \varphi(A_\cap)_i = 0 \ \wedge \ \varphi(A_\cap)_{i-1} = 1 \ \wedge \ \varphi(A_\cap)_{i+1} = 1 \\
\iff & \nexists 1 \leq i \leq k-3 && : \varphi(A_\cap)_i = 1 \ \wedge \ \varphi(A_\cap)_{i+1} = 0 \ \wedge \ \varphi(A_\cap)_{i+2} = 1
\end{aligned}
$$

Also ist $M \subseteq A_\cap$ genau dann dominant, wenn $\varphi(A_\cap)$ die Folge 101 nicht als Teilfolge enthält. Das heißt insgesamt gilt:

$$
\begin{aligned}
|UN(A)| &= |\{M \subseteq A_\cap : M \text{ ist dominant über } A_\cap\}| \\
&= |\{a \in \{0,1\}^{k-1} \mid \nexists 1 \leq i \leq k-3 : a_i = 1 \ \wedge \ a_{i+1} = 0 \ \wedge \ a_{i+2} = 1\}| \\
&= \mu_{k-1}
\end{aligned}
$$

Das ist die Behauptung. □

Mit Blick auf den Hauptsatz 4.3.6 und die Konstruktion des vollständigen Zerlegungsbaumes, mit Hilfe der Algorithmen 4.1 und 4.2, folgt aus dem letzten Lemma sofort:

Folgerung 4.3.11 *Mit μ_k wie in Definition 4.3.9 gilt $boolw(G_k) = \log_2(\mu_{k-1})$ für alle $k \in \mathbb{N}$, $k \geq 3$.*

Tabelle 4.1: Die ersten Glieder der Folge μ_k im Vergleich zu 2^k

$k =$	$\mu_{k-1} =$	$2^k =$	$2^k - \mu_{k-1} =$
3	4	8	4
4	7	16	9
5	12	32	20
6	21	64	43

7	37	128	91
8	65	256	191
9	114	512	398
10	200	1024	824
11	351	2048	1697
12	616	4096	3480
13	1081	8192	7111
14	1897	16384	14487
15	3329	32768	29439
16	5842	65536	59694
17	10252	131072	120820
18	17991	262144	244153
19	31572	524288	492716
20	55405	1048576	993171

Das Lemma 4.3.10 bedeutet also, dass für eine geschlossene Antwort auf die eigentliche Problemstellung nur noch die Folge μ_k zu bestimmen ist. In der abgebildeten Tabelle sind die ersten Glieder (bis $k = 20$) der Folge μ_k abgebildet. Ein vollständiger Zerlegungsbaum, der eine Zeile-für-Zeile Zerlegung wählt, hat offensichtlich Boolsche Weite $\log_2(2^k) = k$, liefert als obere Schranke an die Boolsche Weite von G_k damit $boolw(G_k) \leq k$ und entspricht damit im Wesentlichen der Schranke, die sich auch aus Satz 3.4.20, zusammen mit $cw(G_k) = k + 1$ (siehe Golumbic und Rotics [2000]), ergibt. Als Referenz zum optimalen vollständigen Zerlegungsbaum sind daher noch diese Werte angegeben.

Um dieses Problem nun allgemein zu lösen, wird ein Weg über formale Sprachen gewählt. Eine solche besteht aus Worten über einem **Alphabet** Σ, d.h. einer Menge von Zeichen, den sogenannten **Buchstaben**. Hier wird nur $\Sigma = \{0, 1\}$ relevant sein. Jede beliebige Zeichenfolge $w = w_1 w_2 \dots w_n$ mit Buchstaben $w_i \in \Sigma$ heißt ein **Wort** über Σ. Sind w_1 und w_2 Wörter über Σ, so sind auch $w_1 w_2$ und $w_2 w_1$ Wörter, die auf natürliche Weise aus w_1 und w_2 entstehen, und im Allgemeinen verschieden sind. Ein besonderes Wort ist das **leere Wort** ϵ, d.h. für jedes Wort w über Σ gilt $w\epsilon = \epsilon w = w$. Für eine umfangreichere Einführung in dieses Gebiet sei auf Standardwerke der theoretischen Informatik, wie etwa Hedtstück [2000], verwiesen. Für die hier benötigten Zwecke reichen aber diese Grundlagen aus.

Definition 4.3.12 *Es werden für alle $k \in \mathbb{N}_0$ die folgenden formalen Sprachen über $\Sigma = \{0,1\}$ definiert:*

$$L(k) := \{w \in \{0,1\}^k \mid \forall i,j\ \forall w_1 \in \{0,1\}^i\ \forall w_2 \in \{0,1\}^j : w \neq w_1 101 w_2\}$$

$$L_1(k) := \begin{cases} \emptyset & \text{falls } k = 0 \\ L(k) \cap \{w \in \{0,1\}^k : w_k = 1\} & \text{falls } k \geq 1 \end{cases}$$

$$L_0(k) := \begin{cases} \{\epsilon\} & \text{falls } k = 0 \\ L(k) \cap \{w \in \{0,1\}^k : w_k = 0\} & \text{falls } k \geq 1 \end{cases}$$

L enthält also alle Wörter über Σ der Länge k, die "101" nicht als Teilwort enthalten, und L_i alle Wörter aus L, die auf $i = 0$ bzw. $i = 1$ enden. Eine Ausnahme bildet hier nur der Fall $k = 0$. Einige grundlegende, aber sehr wichtige Eigenschaften, die jeweils sofort aus den Definitionen folgen und insbesondere den Spezialfall für $k = 0$ aufklären, sollen kurz festgehalten werden:

Bemerkung 4.3.13 *Mit $L(k)$, $L_0(k)$, $L_1(k)$ und μ_k wie in den Definitionen 4.3.12 sowie 4.3.9 gilt:*

1. *Für alle $k \in \mathbb{N}$ gilt $\mu_k = |L(k)|$.*
2. *Für alle $k \in \mathbb{N}_0$ gilt $L(k) = L_0(k) \uplus L_1(k)$.*

Nun sollen zwei wichtige Resultate gezeigt werden, auf denen die weitere Analyse aufbaut. Doch vorab eine kurze Definition.

Definition 4.3.14 *Es seien $L(k)$, $L_0(k)$ und $L_1(k)$ wie in Definition 4.3.12. Die Zahlenfolgen A_k, B_k und C_k werden für $k \in \mathbb{N}_0$ wie folgt definiert: $A_k := |L_1(k)|$, $B_k := |L_0(k)|$ und $C_k := |L(k)|$. Für $k < 0$ wird dies der Einfachheit halber verallgemeinert zu $A_k := B_k := C_k := 0$.*

Bemerkung 4.3.13 impliziert dann $C_k = A_k + B_k$ für alle $k \in \mathbb{Z}$ sowie $C_k = \mu_k$ für alle $k \in \mathbb{N}$. Vor diesem Hintergrund sind die folgenden Aussagen zu verstehen.

Lemma 4.3.15 *Für alle $k \in \mathbb{Z}$ gilt $A_k = A_{k-1} + B_{k-1} - A_{k-2}$.*

Beweis
Es gelte $k \geq 2$, denn sonst ist die Behauptung trivial. Für alle $k \geq 2$ gilt nach Definition $A_k = |L_1(k)| = |L(k) \cap \{w \in \{0,1\}^k : w_k = 1\}|$. Es sind also die Wörter der Länge $k-1$ zu untersuchen, die 101 nicht als Teilwort enthalten. Solche gibt es genau $|L(k-1)| = C_{k-1}$ Stück. Zusätzlich darf das Wort aber auch nicht auf '10'

enden, denn sonst entsteht durch die zusätzliche '1' das Teilwort '101' am Ende des Wortes. Es sind also genau $|L(k-1) \cap \{w \in \{0,1\}^{k-1} : w_{k-2} = 1 \ \wedge \ w_{k-1} = 0\}|$ zu subtrahieren. Zusammen folgt also:

$$\begin{aligned} A_k &= |L_1(k)| = |L(k) \cap \{w \in \{0,1\}^k : w_k = 1\}| \\ &= |L(k-1)| - |L(k-1) \cap \{w \in \{0,1\}^{k-1} : w_{k-2} = 1 \ \wedge \ w_{k-1} = 0\}| \\ &= |L(k-1)| - |L_1(k-2)| = C_{k-1} - A_{k-2} = A_{k-1} + B_{k-1} - A_{k-2} \end{aligned}$$

Das ist die Behauptung. □

Eine ähnliche Aussage soll für B_k bewiesen werden. Dazu wird für eine Gleichung $a = b$, mit a und b geeignet gewählt, folgende Notation verwendet:

$$[a = b] := \begin{cases} 1 & \text{wenn } a = b \\ 0 & \text{wenn } a \neq b \end{cases}$$

Lemma 4.3.16 *Für alle $k \in \mathbb{Z}$ gilt $B_k = A_{k-1} + B_{k-1} + [k = 0]$.*

Beweis
Die Fälle $k < 0$ und $k = 0$ gelten trivialerweise bzw. sofort nach Definition von $B_k = |L_0(k)|$, vgl. 4.3.12. Daher sei $k \geq 1$. Es folgt:

$$\begin{aligned} B_k &= |L_0(k)| = |L(k) \cap \{w \in \{0,1\}^k : w_k = 0\}| \\ &= |L(k-1)| = C_{k-1} = A_{k-1} + B_{k-1} = A_{k-1} + B_{k-1} + \underbrace{[k = 0]}_{=0, \text{ für } k \geq 1} \end{aligned}$$

Das ist die Behauptung. □

Nun werden einige Potenzreihen definiert, mit deren Hilfe sich schließlich die Folge C_k, und somit auch μ_k, bestimmen lässt.

Definition 4.3.17 *Die Folgen A_k, B_k und C_k seien wie in Definition 4.3.14 gegeben. Dann werden die formalen Potenzreihen $\mathcal{A}(z)$, $\mathcal{B}(z)$ und $\mathcal{C}(z)$ wie folgt definiert:*

$$\mathcal{A}(z) := \sum_{k=0}^{\infty} A_k \cdot z^k \in \mathbb{R}[[z]]$$

$$\mathcal{B}(z) := \sum_{k=0}^{\infty} B_k \cdot z^k \in \mathbb{R}[[z]]$$

$$\mathcal{C}(z) := \sum_{k=0}^{\infty} C_k \cdot z^k \in \mathbb{R}[[z]]$$

Einige grundlegende Eigenschaften von $\mathcal{A}(z)$, $\mathcal{B}(z)$ und $\mathcal{C}(z)$ sollen kurz festgehalten werden.

Lemma 4.3.18 *Mit der Notation aus den Definitionen 4.3.12, 4.3.14 und 4.3.17 gilt:*

1. $\mathcal{A}(z) = z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) - z^2 \cdot \mathcal{A}(z)$
2. $\mathcal{B}(z) = z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) + 1$
3. $\mathcal{C}(z) = \mathcal{A}(z) + \mathcal{B}(z) = 2 \cdot z \cdot \mathcal{C}(z) + 1 - z^2 \cdot \mathcal{A}(z)$

Beweis

1. Nach Lemma 4.3.15 gilt $A_k = A_{k-1} + B_{k-1} - A_{k-2}$ für alle $k \in \mathbb{Z}$. Also folgt:

$$\begin{aligned}
\mathcal{A}(z) &= \sum_{k=0}^{\infty} A_k \cdot z^k = \sum_{k=0}^{\infty} (A_{k-1} + B_{k-1} - A_{k-2}) \cdot z^k \\
&= \sum_{k=0}^{\infty} A_{k-1} \cdot z^k + \sum_{k=0}^{\infty} B_{k-1} \cdot z^k - \sum_{k=0}^{\infty} A_{k-2} \cdot z^k \\
&= z \cdot \sum_{k=1}^{\infty} A_{k-1} \cdot z^{k-1} + z \cdot \sum_{k=1}^{\infty} B_{k-1} \cdot z^{k-1} - z^2 \cdot \sum_{k=2}^{\infty} A_{k-2} \cdot z^{k-2} \\
&= z \cdot \sum_{k=0}^{\infty} A_k \cdot z^k + z \cdot \sum_{k=0}^{\infty} B_k \cdot z^k - z^2 \cdot \sum_{k=0}^{\infty} A_k \cdot z^k \\
&= z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) - z^2 \cdot \mathcal{A}(z)
\end{aligned}$$

2. Analog ist $B_k = A_{k-1} + B_{k-1} + [k = 0]$ für alle $k \in \mathbb{Z}$, unter Verwendung von Lemma 4.3.16. Damit folgt wieder:

$$\begin{aligned}
\mathcal{B}(z) &= \sum_{k=0}^{\infty} B_k \cdot z^k = \sum_{k=0}^{\infty} (A_{k-1} + B_{k-1} + [k = 0]) \cdot z^k \\
&= \sum_{k=0}^{\infty} A_{k-1} \cdot z^k + \sum_{k=0}^{\infty} B_{k-1} \cdot z^k + \sum_{k=0}^{\infty} [k = 0] \cdot z^k \\
&= z \cdot \sum_{k=1}^{\infty} A_{k-1} \cdot z^{k-1} + z \cdot \sum_{k=1}^{\infty} B_{k-1} \cdot z^{k-1} + 1 \\
&= z \cdot \sum_{k=0}^{\infty} A_k \cdot z^k + z \cdot \sum_{k=0}^{\infty} B_k \cdot z^k + 1 \\
&= z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) + 1
\end{aligned}$$

3. Durch Kombinieren von Punkt 1. und 2. folgt:

$$\begin{aligned}\mathcal{C}(z) &= \sum_{k=0}^{\infty} C_k \cdot z^k = \sum_{k=0}^{\infty} (A_k + B_k) \cdot z^k \\ &= \sum_{k=0}^{\infty} A_k \cdot z^k + \sum_{k=0}^{\infty} B_k \cdot z^k = \mathcal{A}(z) + \mathcal{B}(z) \\ &= (z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) - z^2 \cdot \mathcal{A}(z)) + (z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) + 1) \\ &= 2 \cdot z \cdot (\mathcal{A}(z) + \mathcal{B}(z)) + 1 - z^2 \cdot \mathcal{A}(z) = 2 \cdot z \cdot \mathcal{C}(z) + 1 - z^2 \cdot \mathcal{A}(z)\end{aligned}$$ □

Mit diesen Eigenschaften kann ein erstes wichtiges Resultat zur formalen Potenzreihe $\mathcal{C}(z)$ festgehalten werden. Vorab sei dazu angemerkt, dass sich die weitere Analyse auf viele andere Folgen in gleicher Weise übertragen lässt, sprich die Verfahren im Wesentlichen allgemein gültig sind. Da dies hier allerdings nicht weiter wichtig ist, sei dazu an dieser Stelle auf die Literatur, wie etwa Graham u. a. [1994], Stanley [2004] oder Nörlund [1924], verwiesen.

Lemma 4.3.19 *Es gilt* $\mathcal{C}(z) \cdot (1 - 2 \cdot z + z^2 - z^3) = 1 + z^2$.

Beweis
Nach Lemma 4.3.18 gilt einerseits

$$\mathcal{B}(z) = z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) + 1 \iff \mathcal{A}(z) = \frac{1}{z} \cdot (\mathcal{B}(z) - z \cdot \mathcal{B}(z) - 1)$$

und andererseits

$$\mathcal{B}(z) = z \cdot \mathcal{A}(z) + z \cdot \mathcal{B}(z) + 1 = z \cdot \mathcal{C}(z) + 1.$$

Mit Lemma 4.3.18 3.) folgt daraus:

$$\begin{aligned}\mathcal{C}(z) &= 2 \cdot z \cdot \mathcal{C}(z) + 1 - z^2 \cdot \mathcal{A}(z) \\ &= 2 \cdot z \cdot \mathcal{C}(z) + 1 - z^2 \cdot \left(\frac{1}{z} \cdot (\mathcal{B}(z) - z \cdot \mathcal{B}(z) - 1)\right) \\ &= 2 \cdot z \cdot \mathcal{C}(z) + 1 - z \cdot (1 - z) \cdot \mathcal{B}(z) + z \\ &= 2 \cdot z \cdot \mathcal{C}(z) + 1 - z \cdot (1 - z) \cdot (z \cdot \mathcal{C}(z) + 1) + z \\ &= 2 \cdot z \cdot \mathcal{C}(z) - z^2 \cdot (1 - z) \cdot \mathcal{C}(z) - z \cdot (1 - z) + 1 + z \\ &= \mathcal{C}(z) \cdot z \cdot (2 - z \cdot (1 - z)) + 1 + z^2 \\ &= \mathcal{C}(z) \cdot z \cdot (2 - z + z^2) + 1 + z^2\end{aligned}$$

Es gilt hier ferner:

$$\begin{aligned} & \mathcal{C}(z) = \mathcal{C}(z) \cdot z \cdot (2 - z + z^2) + 1 + z^2 \\ \iff & \mathcal{C}(z) \cdot \left(1 - z \cdot (2 - z + z^2)\right) = 1 + z^2 \\ \iff & \mathcal{C}(z) \cdot \left(1 - 2 \cdot z + z^2 - z^3\right) = 1 + z^2 \end{aligned}$$

Das ist die Behauptung. □

Durch geschicktes Umformulieren des Ausdrucks $1 + z^2$ folgt aus Lemma 4.3.19 eine Rekursionsgleichung für die Folge C_k:

Folgerung 4.3.20 *Für alle $k \in \mathbb{Z}$ erfüllt die Folge C_k aus Definition 4.3.14 die Rekursion:*

$$C_k = 2 \cdot C_{k-1} - C_{k-2} + C_{k-3} + [k = 0] + [k = 2]$$

Mit μ_k wie in Definition 4.3.9 gilt nach Bemerkung 4.3.13 also insbesondere

$$\mu_k = 2 \cdot \mu_{k-1} - \mu_{k-2} + \mu_{k-3}$$

für alle $k \in \mathbb{N}$, $k \geq 4$.

Beweis
Nach Definition ist $\mathcal{C}(k) = \sum_{k=0}^{\infty} C_k z^k$. Unter Verwendung von $C_k = 0$ für $k < 0$ und mit Lemma 4.3.19 folgt daher:

$$\begin{aligned} 0 &= \mathcal{C}(z) \cdot \left(1 - 2 \cdot z + z^2 - z^3\right) - (1 + z^2) \\ &= \left(\sum_{k=0}^{\infty} C_k \cdot z^k\right) \cdot \left(1 - 2 \cdot z + z^2 - z^3\right) - \left(\sum_{k=0}^{\infty} ([k = 0]) \cdot z^k + \sum_{k=0}^{\infty} ([k = 2]) \cdot z^k\right) \\ &= \sum_{k=0}^{\infty} \left((1 - 2 \cdot z + z^2 - z^3) \cdot C_k \cdot z^k\right) - \sum_{k=0}^{\infty} ([k = 0] + [k = 2]) \cdot z^k \\ &= \sum_{k=0}^{\infty} (C_k - [k = 0] - [k = 2]) \cdot z^k - 2 \cdot \sum_{k=0}^{\infty} C_k \cdot z^{k+1} + \sum_{k=0}^{\infty} C_k \cdot z^{k+2} - \sum_{k=0}^{\infty} C_k \cdot z^{k+3} \\ &= \sum_{k=0}^{\infty} (C_k - [k = 0] - [k = 2]) \cdot z^k - 2 \cdot \sum_{k=1}^{\infty} C_{k-1} \cdot z^k + \sum_{k=2}^{\infty} C_{k-2} \cdot z^k - \sum_{k=3}^{\infty} C_{k-3} \cdot z^k \\ &= \sum_{k=0}^{\infty} (C_k - [k = 0] - [k = 2]) \cdot z^k - 2 \cdot \sum_{k=0}^{\infty} C_{k-1} \cdot z^k + \sum_{k=0}^{\infty} C_{k-2} \cdot z^k - \sum_{k=0}^{\infty} C_{k-3} \cdot z^k \\ &= \sum_{k=0}^{\infty} (C_k - 2 \cdot C_{k-1} + C_{k-2} - C_{k-3} - [k = 0] - [k = 2]) \cdot z^k \end{aligned}$$

Es folgt also $C_k - 2 \cdot C_{k-1} + C_{k-2} - C_{k-3} - [k=0] - [k=2] = 0$ und damit sofort $C_k = 2 \cdot C_{k-1} - C_{k-2} + C_{k-3} + [k=0] + [k=2]$. Das ist die Behauptung. □

Mit diesen Hilfsmitteln ist es also möglich den Wert μ_k, für festes k, zu bestimmen. Verbleibende Ziele sind es dann, noch eine asymptotische Abschätzung für die Folge μ_k herzuleiten, um diese schließlich mit dem Hauptsatz 4.3.6 auf die Boolsche Weite von Gittergraphen zu übertragen. Doch zunächst eine einfach zu zeigende, aber dennoch wichtige Eigenschaft zur Folge μ_k:

Folgerung 4.3.21 *Die Folge μ_k aus Definition 4.3.9 ist für alle $k \in \mathbb{N}$ streng monoton wachsend.*

Beweis
Direktes Nachrechnen zeigt, dass $\mu_1 = 2$, $\mu_2 = 4$, $\mu_3 = 7$ und $\mu_4 = 12$ (siehe dazu auch Tabelle 4.1). Die Behauptung gilt also für $k \leq 4$. Ansonsten liefert Folgerung 4.3.20 die Identität $\mu_k = 2 \cdot \mu_{k-1} - \mu_{k-2} + \mu_{k-3}$. Es folgt per vollständiger Induktion:

$$\begin{aligned}\mu_k &= 2 \cdot \mu_{k-1} - \mu_{k-2} + \mu_{k-3} = \mu_{k-1} + \overbrace{\mu_{k-1} - \mu_{k-2}}^{>0 \text{ nach (IV)}} + \mu_{k-3} \\ &> \mu_{k-1} + \mu_{k-3} \geq \mu_{k-1}\end{aligned}$$

□

Damit ist im Wesentlichen alles für das zweite Hauptergebnis dieses Kapitels vorbereitet, das die Frage nach der Boolschen Weite der Gittergraphen G_k beantwortet. Begonnen werden soll allerdings noch mit einer kurzen Analyse der Gleichung $\mathcal{C}(z) \cdot (1 - 2 \cdot z + z^2 - z^3) = 1 + z^2$. Diese lässt sich für $1 - 2 \cdot z + z^2 - z^3 \neq 0$ äquivalent reformulieren zu:

$$\mathcal{C}(z) = \frac{1 + z^2}{1 - 2 \cdot z + z^2 - z^3}$$

Es seien λ_1, λ_2 und λ_3 die Nullstellen des Nennerpolynoms $1 - 2 \cdot z + z^2 - z^3$, wobei $\lambda_1 \approx 0.56984$ reell ist und $\lambda_2 \approx 0.21508 + 1.30714 \cdot i$ sowie $\lambda_2 \approx 0.21508 - 1.30714 \cdot i$ komplex sind. Mit einer Partialbruchzerlegung folgt die Existenz von Konstanten α_1, α_2 und α_3, so dass

$$\mathcal{C}(z) = \frac{1 + z^2}{1 - 2 \cdot z + z^2 - z^3} = \frac{\alpha_1}{z - \lambda_1} + \frac{\alpha_2}{z - \lambda_2} + \frac{\alpha_3}{z - \lambda_3}$$

gilt.

Hauptsatz 4.3.22 *Die Boolsche Weite der Gittergraphen G_k ist für alle $k \geq 3$*

$$boolw(G_k) = \log_2(\mu_{k-1}) = \log_2\left(-\alpha_1 \cdot \left(\frac{1}{\lambda_1}\right)^k + \xi(k)\right)$$

wobei der Korrekturterm $\xi(k) \overset{k\to\infty}{\longrightarrow} 0$ *konvergiert und* μ_k *wie in Definition 4.3.9 ist. Die Konstante* λ_1 *ist dabei die eindeutige reelle Nullstelle der Polynomfunktion* $Q(z) = 1 - 2 \cdot z + z^2 - z^3$ *und* α_1 *der eindeutige Koeffizient von* $\frac{1}{z-\lambda_1}$ *in einer Partialbruchzerlegung von* $\mathcal{C}(z)$*.*

Beweis

Folgerung 4.3.11 liefert $boolw(G_k) = \log_2(\mu_{k-1})$, so dass $\mu_k = -\alpha_1 \cdot \left(\frac{1}{\lambda_1}\right)^k + \xi(k)$ mit $\xi(k) \overset{k\to\infty}{\longrightarrow} 0$ zu zeigen bleibt. Die Identität $\mathcal{C}(z) = \frac{\alpha_1}{z-\lambda_1} + \frac{\alpha_2}{z-\lambda_2} + \frac{\alpha_3}{z-\lambda_3}$ wurde bereits mit Hilfe einer Partialbruchzerlegung gezeigt, wobei die Konstanten α_i eindeutig bestimmt sind. Also ergibt sich:

$$\begin{aligned}
\mathcal{C}(z) &= \frac{\alpha_1}{z-\lambda_1} + \frac{\alpha_2}{z-\lambda_2} + \frac{\alpha_3}{z-\lambda_3} \\
&= -\frac{\alpha_1}{\lambda_1} \cdot \frac{1}{1-\frac{z}{\lambda_1}} - \frac{\alpha_2}{\lambda_2} \cdot \frac{1}{1-\frac{z}{\lambda_2}} - \frac{\alpha_3}{\lambda_3} \cdot \frac{1}{1-\frac{z}{\lambda_3}} \\
&= -\frac{\alpha_1}{\lambda_1} \cdot \sum_{k=0}^{\infty} \left(\frac{z}{\lambda_1}\right)^k - \frac{\alpha_2}{\lambda_2} \cdot \sum_{k=0}^{\infty} \left(\frac{z}{\lambda_2}\right)^k - \frac{\alpha_3}{\lambda_3} \cdot \sum_{k=0}^{\infty} \left(\frac{z}{\lambda_3}\right)^k \\
&= \sum_{k=0}^{\infty} \left(-\frac{\alpha_1}{\lambda_1} \cdot \left(\frac{1}{\lambda_1}\right)^k - \frac{\alpha_2}{\lambda_2} \cdot \left(\frac{1}{\lambda_2}\right)^k - \frac{\alpha_3}{\lambda_3} \cdot \left(\frac{1}{\lambda_3}\right)^k \right) \cdot z^k \\
&= \sum_{k=0}^{\infty} C_k \cdot z^k \qquad \text{(vgl. Definition 4.3.17)}
\end{aligned}$$

Also folgt für alle $k \in \mathbb{N}$, $k \geq 3$:

$$-\frac{\alpha_1}{\lambda_1} \cdot \left(\frac{1}{\lambda_1}\right)^k - \frac{\alpha_2}{\lambda_2} \cdot \left(\frac{1}{\lambda_2}\right)^k - \frac{\alpha_3}{\lambda_3} \cdot \left(\frac{1}{\lambda_3}\right)^k = C_k \overset{4.3.14}{=} |L(k)| \overset{4.3.13}{=} \mu_k$$

Wird $\xi(k) := -\alpha_2 \cdot \left(\frac{1}{\lambda_2}\right)^k - \alpha_3 \cdot \left(\frac{1}{\lambda_3}\right)^k$ definiert, gilt daher:

$$\begin{aligned}
\mu_{k-1} &= -\frac{\alpha_1}{\lambda_1} \cdot \left(\frac{1}{\lambda_1}\right)^{k-1} - \frac{\alpha_2}{\lambda_2} \cdot \left(\frac{1}{\lambda_2}\right)^{k-1} - \frac{\alpha_3}{\lambda_3} \cdot \left(\frac{1}{\lambda_3}\right)^{k-1} \\
&= -\alpha_1 \cdot \left(\frac{1}{\lambda_1}\right)^k - \alpha_2 \cdot \left(\frac{1}{\lambda_2}\right)^k - \alpha_3 \cdot \left(\frac{1}{\lambda_3}\right)^k \\
&= -\alpha_1 \cdot \left(\frac{1}{\lambda_1}\right)^k + \xi(k)
\end{aligned}$$

Es bleibt also zu zeigen, dass $\lim_{k\to\infty} \xi(k) = 0$ gilt. Direktes Nachrechnen von λ_2 und λ_3 liefert $\lambda_2 \approx 0.21508 + 1.30714 \cdot i$ sowie $\lambda_3 \approx 0.21508 - 1.30714 \cdot i$. Daraus folgt $|\lambda_2| = |\lambda_3| \approx 1.32472$ und somit ist $\left|\frac{1}{\lambda_2}\right| = \left|\frac{1}{\lambda_3}\right| < 1$. Also gilt

$$\lim_{k\to\infty} \left|\frac{1}{\lambda_2}\right|^k = \lim_{k\to\infty} \left|\frac{1}{\lambda_3}\right|^k = 0$$

woraus insbesondere

$$\lim_{k\to\infty} \left(\frac{1}{\lambda_2}\right)^k = \lim_{k\to\infty} \left(\frac{1}{\lambda_3}\right)^k = 0$$

und damit $\lim_{k\to\infty} \xi(k) = 0$ folgt. Das ist die Behauptung. □

Der Hauptsatz 4.3.22 besagt also, dass für hinreichend große k gilt:

$$\begin{aligned} boolw(G_k) &\approx \log_2\left(-\alpha_1 \cdot \left(\frac{1}{\lambda_1}\right)^k\right) \\ &= \log_2\left(\left(\frac{1}{\lambda_1}\right)^k\right) + \log_2(-\alpha_1) \\ &= k \cdot \log_2\left(\frac{1}{\lambda_1}\right) + \log_2(-\alpha_1) \end{aligned}$$

Direktes Berechnen der Konstanten α_1 und λ_1 zeigt, dass $\log_2(-\alpha_1) \approx -0.46968$ und $\log_2\left(\frac{1}{\lambda_1}\right) \approx 0.81137$ ist, also

$$boolw(G_k) \approx 0.81137 \cdot k - 0.46968$$

gilt. Mit $\nu(k) := -\frac{\alpha_1}{\lambda_1} \cdot \left(\frac{1}{\lambda_1}\right)^k \approx \mu_k$ sind hier als Referenz die ersten 20 Werte von $\nu(k)$ und μ_k sowie von $\log_2(\nu(k))$ und $\log_2(\mu_k)$ angegeben. Bei der Bewertung ist allerdings zu beachten, dass hier außer dem Approximationsfehler auch noch Rundungsfehler auftreten.

Tabelle 4.2: Die ersten 20 Glieder der Folge μ_k im Vergleich zu $\nu(k)$

$k =$	$\mu_k =$	$\nu(k) =$	$\log_2(\mu_k) =$	$\log_2(\nu(k)) =$
1	2	2.22385	1	1.15306
2	4	3.90259	2	1.96443
3	7	6.84856	2.80736	2.77580
4	12	12.0184	3.58497	3.58717
5	21	21.0908	4.39232	4.39854
6	37	37.0118	5.20946	5.20991

7	65	64.9511	6.02237	6.02128
8	114	113.981	6.83289	6.83265
9	200	200.023	7.64386	7.64402
10	351	351.016	8.45533	8.45539
11	616	615.991	9.26679	9.26677
12	1081	1080.99	10.0782	10.0781
13	1897	1897.00	10.8895	10.8895
14	3329	3329.01	11.7009	11.7009
15	5842	5842.00	12.5123	12.5122
16	10252	10252.0	13.3236	13.3236
17	17991	17991.0	14.1350	14.1350
18	31572	31572.0	14.9463	14.9464
19	55405	55405.0	15.7577	15.7577
20	97229	97229.0	16.5691	16.5691

Abschließend sollen nun die Ergebnisse dieses Abschnitts noch auf die allgemeineren $(k \times \ell)$-Gittergraphen ausgeweitet werden.

4.3.4 Verallgemeinerung auf $(k \times \ell)$-Gittergraphen

Die $(k \times k)$-Gittergraphen besitzen eine einfache Verallgemeinerung, die nicht aus genau k Zeilen und k Spalten besteht, sondern statt dessen k Zeilen und ℓ Spalten hat. Durch offensichtliche Isomorphie kann ohne Einschränkung $k \leq \ell$ angenommen werden. Formal liefert das folgende Definition, als Verallgemeinerung von Definition 3.4.14.

Definition 4.3.23 *Es seien $k, \ell \in \mathbb{N}$. Der Graph $G_{k\times\ell} = (V, E)$ mit*

- $V = \{v_{i,j} : 1 \leq i \leq k,\ 1 \leq j \leq \ell\}$
- $E = E_1 \uplus E_2$ *wobei*
  - $E_1 = \{\{v_{i,j},\ v_{i,j+1}\} : 1 \leq i \leq k,\ 1 \leq j \leq \ell - 1\}$
  - $E_2 = \{\{v_{i,j},\ v_{i+1,j}\} : 1 \leq i \leq k - 1,\ 1 \leq j \leq \ell\}$

*heißt $(k \times \ell)$-**Gitter**.*

Ist $k = \ell$, so stimmt dies mit Definition 3.4.14 überein. Für alle $k, \ell \in \mathbb{N}$ mit $k \leq \ell$ ist offensichtlich G_k ein induzierter Teilgraph von $G_{k\times\ell}$. Dann folgt aus Satz 3.4.17 sofort:

Folgerung 4.3.24 *Für alle $k, \ell \in \mathbb{N}$ mit $k \leq \ell$ ist $boolw(G_k) \leq boolw(G_{k\times\ell})$.*

Es soll nun noch $boolw(G_{k+1}) \geq boolw(G_{k\times\ell})$ gezeigt werden um die Hauptsätze 4.3.6 und 4.3.22, mit gewissen Einschränkungen, auf $(k\times\ell)$-Gitter verallgemeinern zu können. Dazu bietet es sich an, einen vollständigen Zerlegungsbaum zu konstruieren, der das Gewünschte erfüllt.

Lemma 4.3.25 *Für alle $k, \ell \in \mathbb{N}$ mit $k \leq \ell$ ist $boolw(G_{k+1}) \geq boolw(G_{k\times\ell})$.*

Beweis
Der Beweis ist konstruktiv und orientiert sich an den Algorithmen 4.1 und 4.2, die dort zu $boolw(G_k) = \log_2(\mu_{k-1})$ geführt hatten. Nach Definition gilt, dass mit

$$\begin{aligned} V_1 &= \{\, v_{i,j} \in V(G_{k\times\ell}) : i+j \leq k \quad \} \\ V_2 &= \{\, v_{i,j} \in V(G_{k\times\ell}) : i+j \geq \ell+2 \,\} \\ D_d &= \{\, v_{i,j} \in V(G_{k\times\ell}) : i+j = k+d \,\} \text{ für alle } d = 1,\ldots,\ell-k+1 \end{aligned}$$

die Knotenmenge $V(G_{k\times\ell})$ die disjunkte Vereinigung dieser Mengen ist, d.h. es gilt:

$$V(G_{k\times\ell}) = V_1 \uplus V_2 \uplus \biguplus_{d=1}^{\ell-k+1} D_d$$

Anschaulich bedeutet dies, dass V_1 das obere linke Dreieck des Gitters ist, bis zur ersten Diagonalen der Länge k. Analog ist V_2 das rechte untere Dreieck unterhalb der letzten Diagonalen. Die Mengen D_1 bis $D_{\ell-k+1}$ entsprechen dann genau den $\ell-k+1$ parallelen Diagonalen im Gitter. Zur Konstruktion eines vollständigen Zerlegungsbaumes (T,δ) für $G_{k\times\ell}$ gehe nun wie folgt vor: Beginne mit einem Wurzelknoten r mit $\delta(r) = V(G_{k\times\ell})$ und Söhnen $s_1(r)$ sowie $s_2(r)$, mit Labels $\delta(s_1(r)) = V_1$ bzw. $\delta(s_2(r)) = \overline{V_1}$. Beginne dann mit $t := s_2(r)$ und wähle in jedem Schritt

$$i = \min\{j \in \{1,\ldots,\ell-k+1\} : D_j \subseteq \delta(t)\}$$

und labele anschließend die Söhne $s_1(t)$ bzw. $s_2(t)$ jeweils mit $\delta(s_1(t)) = D_i$ sowie $\delta(s_2(t)) = \delta(t)\backslash D_i$. Bis $\{j \in \{1,\ldots,\ell-k+1\} : D_j \subseteq \delta(t)\} = \emptyset$ gilt, fahre dann mit $t := s_2(t)$ fort. Gehe anschließend wie in Algorithmus 4.1 vor.

Die Mengen $\delta(t)$, die $|UN(\delta(t))|$ über T maximieren, sind dann gerade die Diagonalen $D_1,\ldots,D_{\ell-k+1}$. Für den so konstruierten vollständigen Zerlegungsbaum (T,δ) gilt also $boolw(T,\delta) \leq \log_2(\mu_k) = boolw(G_{k+1})$, in völlig analoger Weise wie für die Gittergraphen G_{k+1}, wobei μ_k wie in Defintion 4.3.9 ist. Insgesamt folgt die Behauptung. □

Folgerung 4.3.24 und Lemma 4.3.25 implizieren gemeinsam sofort das nächste Hauptergebnis.

Hauptsatz 4.3.26 *Für die Boolsche Weite der $(k\times\ell)$-Gittergraphen $G_{k\times\ell}$ mit $k \leq \ell$ gilt $boolw(G_k) \leq boolw(G_{k\times\ell}) \leq boolw(G_{k+1})$.*

4.4 Analyse der H-Gitter

In Kapitel 3 ist eine weitere Klasse von Graphen aufgetreten, die dort dazu verwendet wurde zu zeigen, dass es eine unendliche Familie von Graphen, mit beliebig großer Boolscher Weite gibt, die beliebig viele und beliebig große Teilgraphen bzw. Minoren haben, deren Boolsche Weite konstant ist, während es ihre Baumweite nicht ist. Dazu hatte die Abschätzung $boolw(HG_k) \geq \log_2(k) - 1$ aus Folgerung 3.4.24 genügt, die sich aus der Cliquenweite $k \leq cw(HG_k) \leq k+1$ (vgl. Satz 3.4.23) und der allgemein gültigen Aussage $\log_2(cw(G)) - 1 \leq boolw(G)$, siehe Satz 3.4.20, ergibt. Daher soll nun noch die Boolsche Weite dieser Graphklasse möglichst genau bestimmt werden. Zunächst sei dazu an die Definition 3.4.21 erinnert:

Ein H-Gitter ist für $k \in \mathbb{N}$ definiert durch

- $V := \{v_{i,j} : 1 \leq i, j \leq k\}$
- $E := E_1 \uplus E_2 \uplus E_3$ wobei
  - $E_1 = \{\{v_{i_1,j},\ v_{i_2,j}\} : 1 \leq i_1 \leq k,\ 1 \leq i_2 \leq k,\ i_1 \neq i_2,\ 1 \leq j \leq k\}$
  - $E_2 = \{\{v_{i_1,j},\ v_{i_2,j+1}\} : j \in 2\cdot\mathbb{N}-1,\ j \leq k-1,\ 2 \leq i_1 \leq k,\ 1 \leq i_2 \leq i_1-1\}$
  - $E_3 = \{\{v_{i_1,j},\ v_{i_2,j+1}\} : j \in 2\cdot\mathbb{N},\ j \leq k-1,\ 1 \leq i_1 \leq k-1,\ i_1+1 \leq i_2 \leq k\}$

und wird mit $HG_k := (V, E)$ bezeichnet. In Kapitel 3 sind die H-Gitter HG_3, HG_4 und HG_5 abgebildet, siehe Abbildung 3.3.

Nun soll konstruktiv eine obere Schranke an die Boolsche Weite der H-Gitter gezeigt werden, die im Wesentlichen der unteren Schranke $boolw(HG_k) \geq \log_2(k) - 1$ aus Folgerung 3.4.24 entspricht.

Lemma 4.4.1 *Für alle $k \in \mathbb{N}$ gilt $boolw(HG_k) \leq \log_2(k)$.*

Beweis
Die Behauptung folgt, wenn ein vollständiger Zerlegungsbaum (T, δ) gefunden wird, der $boolw(HG_k) \leq \log_2(k)$ erfüllt. Für $k = 1$ ist nichts zu zeigen, also gelte $k \geq 2$.

Nun muss $\max_{t \in T} |UN(\delta(t))| \leq k$ gelten. Es reicht also aus, $|UN(\delta(t))| \leq k$ für alle $t \in T$ zu zeigen. Dazu sei für $1 \leq \ell \leq k$ die Menge

$$S_\ell := \{v_{i,\ell} \in V(HG_k) : 1 \leq i \leq k\},$$

d.h. S_ℓ ist die ℓ-te Spalte des Graphen HG_k. Konstruiere nun einen vollständigen Zerlegungsbaum (T, δ) wie folgt: Definiere einen Wurzelknoten r mit $\delta(r) := V(HG_k)$ und bestimme die Menge S_i mit $i = i(t) = \min\{j \in \{1, \ldots, k-1\} : S_j \subseteq \delta(t)\}$, beginnend bei $t = r$. Füge dann die Söhne $s_1 = s_1(t)$ sowie $s_2 = s_2(t)$ mit $\delta(s_1) = S_i$ und $\delta(s_2) := \delta(t) \backslash S_i$ zu T hinzu.

Dieses Vorgehen konstruiert also einen Zerlegungsbaum, bei dem sukzessive die Spalten S_ℓ aus $V(HG_k)$ entfernt werden. Nun sind noch die Spalten zu zerlegen. Diese bilden jeweils eine Clique, können also ebenfalls durch sukzessives Entfernen von einem Knoten so zerlegt werden, dass für die so entstehenden Mengen A immer $|UN(A)| \leq \max_{t \in T} |UN(\delta(t))|$ gilt.

Es verbleibt also zu zeigen, dass $|UN(S_\ell)| = k$ für jede der Spalten S_ℓ mit $1 \leq \ell \leq k$ gilt. Auch hier soll dies wieder über dominante Mengen gezeigt werden, d.h. unter Verwendung von Folgerung 4.2.5, die dann besagt:

$$|UN(S_\ell)| = |\{M \subseteq S_\ell \cap N(\overline{S_\ell}) : M \text{ ist dominant über } S_\ell\}|$$

Nun sei S_ℓ beliebig, wobei ℓ zunächst als ungerade angenommen ist, sowie $M \subseteq S_\ell$ dominant und nicht-leer. Weiter sei

$$z = \max\{i \in \{2, \ldots, k\} : v_{i,\ell} \in S_\ell\},$$

d.h. z ist der größte Zeilenindex, der in M vorkommt, und nach Voraussetzung an M existiert (zu beachten ist hier, dass $N(v_{1,\ell}) \cap \overline{S_\ell} = \emptyset$ gilt). Für alle $1 < i < z$ gilt dann nach Definition $N(v_{i,\ell}) \subset N(v_{z,\ell})$, also folgt $M = \{v_{2,\ell}, \ldots, v_{z,\ell}\} =: S_{\ell,z}$ aus Lemma 4.2.2. Damit sind genau die Mengen $\emptyset, S_{\ell,2}, \ldots, S_{\ell,k}$ über S_ℓ dominant und es folgt $|UN(S_\ell)| = k$.

Nun verbleibt also noch der Fall, dass ℓ ein gerader Spaltenindex ist. Wähle wieder ein dominantes und nicht-leeres $M \subseteq S_\ell$. Hier sei nun

$$z = \min\{i \in \{1, \ldots, k-1\} : v_{i,\ell} \in S_\ell\},$$

d.h. diesmal ist z der kleinste Zeilenindex, der in M vorkommt, und ebenfalls nach Voraussetzung an M existiert. Es folgt $N(v_{i,\ell}) \subset N(v_{z,\ell})$ für alle $z < i < k$, wo-

mit sich $M = \{v_{z,\ell}, \ldots, v_{k-1,\ell}\} =: S_{\ell,z}$, erneut aus Lemma 4.2.2, ergibt. Damit sind also genau die Mengen $\emptyset, S_{\ell,1}, \ldots, S_{\ell,k-1}$ über S_ℓ dominant und es folgt wieder $|UN(S_\ell)| = k$.

Werden alle Teilergebnisse zusammengesetzt, ist (T, δ) ein vollständiger Zerlegungsbaum für HG_k mit $boolw(T, \delta) = \log_2(k)$. Es folgt daher $boolw(HG_k) \leq \log_2(k)$, was zu zeigen war. □

Lemma 4.4.1 und Folgerung 3.4.24 implizieren gemeinsam sofort:

Hauptsatz 4.4.2 *Für die Boolsche Weite der H-Gitter HG_k gilt:*

$$\log_2(k) - 1 \leq boolw(HG_k) \leq \log_2(k)$$

Es drängt sich dabei die Vermutung auf, dass $boolw(HG_k) = \log_2(k)$ gilt. Um dies zu zeigen - insofern die Vermutung zutrifft - müsste die untere Schranke von $boolw(HG_k) \geq \log_2(k) - 1$ zu $boolw(HG_k) \geq \log_2(k)$ verbessert werden. Ein mögliches Werkzeug hierzu wäre der Satz 4.3.1, der schon für die untere Schranke an die Boolsche Weite der Gittergraphen G_k entscheidend war.

Interessant ist dieses Ergebnis aber inbesondere auch mit Blick auf die Aussage aus Satz 3.4.20, wonach $\log(cw(G)) - 1 \leq boolw(G) \leq cw(G)$ stets erfüllt ist. Einerseits gilt nämlich $k \leq cw(HG_k) \leq k + 1$ nach Satz 3.4.23, nach Hauptsatz 4.4.2 andererseits $\log_2(k) - 1 \leq boolw(HG_k) \leq \log_2(k)$. Die H-Gitter treffen also fast genau die untere Schranke aus Satz 3.4.20.

4.5 Analyse der I-Gitter

In Golumbic und Rotics [2000] wurde, neben den $(k \times k)$- bzw. $(k \times \ell)$-Gittergraphen und den H-Gittern HG_k, auch die Cliquenweite der I-Gitter analysiert. Daher soll hier ebenfalls die Boolsche Weite dieser untersucht werden.

Definition 4.5.1 (Golumbic & Rotics) *Es sei $k \in \mathbb{N}$. Der Graph $IG_k = (V, E)$ heißt ein **I-Gitter** der Ordnung k^2 und wird wie folgt definiert:*

- $V := \{v_{i,j} : 1 \leq i, j \leq k\}$
- $E := E_1 \uplus E_2$ *wobei*
  - $E_1 = \{\{v_{i_1,j},\ v_{i_2,j}\} : 1 \leq i_1 \leq k,\ 1 \leq i_2 \leq k,\ i_1 \neq i_2,\ 1 \leq j \leq k\}$
  - $E_2 = \{\{v_{i_1,j},\ v_{i_2,j+1}\} : 1 \leq j \leq k-1,\ 2 \leq i_1 \leq k,\ 1 \leq i_2 \leq i_1 - 1\}$

Etwas informell bedeutet diese Definition, dass die I-Gitter ebenfalls Gittergraphen auf k Zeilen und k Spalten sind, so dass einerseits durch E_1 jede Spalte eine k-Clique bildet (wie auch bei den H-Gittern) und anderseits durch E_2 ein Knoten $v_{i,j}$ mit allen $v_{\ell,j-1}$ mit $\ell < i$ benachbart ist. Abbildung 4.10 zeigt die I-Gitter IG_3, IG_4 und IG_5.

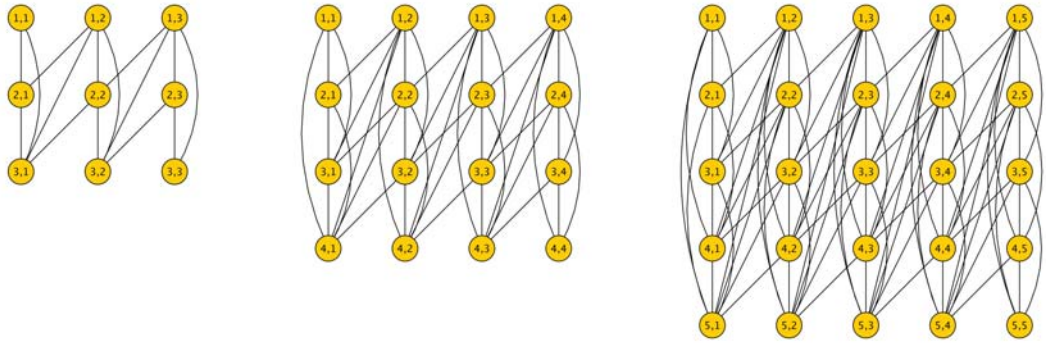

Abbildung 4.10: Die I-Gitter IG_3, IG_4 und IG_5

Ebenfalls wurde in Golumbic und Rotics [2000] folgende wichtige Aussage gezeigt, die die Cliquenweite der I-Gitter betrifft.

Satz 4.5.2 (Golumbic und Rotics [2000]) *Für die Cliquenweite der I-Gitter gilt* $k \leq cw(IG_k) \leq k+1$ *für alle* $k \in \mathbb{N}$.

Nach Satz 3.4.20 gilt $\log_2(cw(G)) - 1 \leq boolw(G) \leq cw(G)$ für jeden Graphen G. Also ergibt Satz 4.5.2 sofort die folgende Aussage.

Folgerung 4.5.3 *Für alle* $k \in \mathbb{N}$ *gilt* $\log_2(k) - 1 \leq boolw(IG_k) \leq k+1$.

Die obere Schranke ist allerdings noch relativ schlecht. Durch die Konstruktion eines geeigneten vollständigen Zerlegungsbaumes liefert das folgende Lemma eine wesentlich bessere Abschätzung.

Lemma 4.5.4 *Für alle* $k \in \mathbb{N}$, $k \geq 2$ *gilt:*

$$boolw(IG_k) \leq \log_2\left(\frac{k \cdot (k-1)}{2} + 1\right) \leq 2 \cdot \log_2(k) - 1$$

Beweis

Der Beweis ist konstruktiv und im Grunde analog zu Lemma 4.4.1 zu sehen, lediglich führt die Konstruktion des Zerlegungsbaumes hier zu $|UN(\delta(t))| \leq \frac{k\cdot(k-1)}{2} + 1$, während für die H-Gitter die Abschätzung um den Faktor $\frac{(k-1)}{2}$ besser war.

Auch hier folgt die Behauptung, wenn ein vollständiger Zerlegungsbaum (T, δ) gefunden wird, der $boolw(IG_k) \leq \log_2\left(\frac{k\cdot(k-1)}{2}+1\right)$ erfüllt. Dazu muss nach Definition $\max\limits_{t\in T} |UN(\delta(t))| \leq \left(\frac{k\cdot(k-1)}{2}+1\right)$, also $|UN(\delta(t))| \leq \left(\frac{k\cdot(k-1)}{2}+1\right)$ für alle $t \in T$, gelten. Wie im Beweis von Lemma 4.4.1 sei für $1 \leq \ell \leq k$ die Menge S_ℓ als

$$S_\ell := \{v_{i,\ell} \in V(HG_k) : 1 \leq i \leq k\},$$

definiert, S_ℓ ist also die ℓ-te Spalte des Graphen IG_k. Konstruiere damit nun einen vollständigen Zerlegungsbaum (T, δ) durch sukzessives Entfernen der Spalten, wie bei den H-Gittern: Definiere einen Wurzelknoten r mit $\delta(r) := V(IG_k)$ und bestimme die Menge S_i mit $i = i(t) = \min\{j \in \{1, \ldots, k-1\} : S_j \subseteq \delta(t)\}$, ausgehend von $t = r$. Konstruiere dann die Söhne $s_1 = s_1(t)$ sowie $s_2 = s_2(t)$ mit $\delta(s_1) = S_i$ und $\delta(s_2) := \delta(t) \backslash S_i$.

Die Spalten können nun beliebig zerlegt werden, denn die maximierenden Mengen bezüglich $|UN(\delta(t))|$ sind dann genau die Spalten S_ℓ mit $2 \leq \ell \leq k-1$. Die Spalten S_1 und S_k liefern, über die selbe Argumentation wie im Beweis von Lemma 4.4.1, gerade $|UN(S_1)| = |UN(S_\ell)| = k \leq \left(\frac{k\cdot(k-1)}{2}+1\right)$. Es verbleibt diese Abschätzung also für S_ℓ mit $1 < \ell < k$ zu zeigen, was auch hier über die Strategie der dominanten Mengen getan wird. Mit Folgerung 4.2.5 gilt:

$$|UN(S_\ell)| = |\{M \subseteq S_\ell \cap N(\overline{S_\ell}) : M \text{ ist dominant über } S_\ell\}|$$

Nun sei $2 \leq \ell \leq k-1$ und $\emptyset \neq M \subseteq S_\ell$ beliebig und dominant. Betrachte nun folgende Zeilenindizes:

$$z_1 := \min\ \{i \in \{1, \ldots, k\} : v_{i,\ell} \in S_\ell\}$$
$$z_2 := \max\ \{i \in \{1, \ldots, k\} : v_{i,\ell} \in S_\ell\}$$

Also z_1 ist der "oberste" Knoten aus Spalte ℓ, der in M enthalten ist, und z_2 der "unterste". Da $M \neq \emptyset$ ist, existieren beide Werte. Da $N(v_{i,\ell}) \subset N(\{v_{z_1,\ell}, v_{z_2,\ell}\})$ für alle $z_1 \leq i \leq z_2$ nach Definition ist, folgt $M = \{v_{i,\ell} : z_1 \leq i \leq z_2\} =: S_{\ell,z_1,z_2}$ aus Lemma 4.2.2.

Die über S_ℓ dominanten Mengen sind damit die leere Menge $\emptyset$ und alle S_{ℓ,z_1,z_2}, für die $1 \leq z_1 < z_2 \leq k$ gilt. Dies sind insgesamt, unabhängig von ℓ, genau $\binom{k}{2} = \frac{k\cdot(k-1)}{2}$ viele. Wird die leere Menge hinzugenommen, folgt insgesamt $|UN(S_\ell)| = \frac{k\cdot(k-1)}{2}+1$.

Werden nun alle Teilergebnisse zusammengesetzt, ergibt sich ein vollständiger Zerlegungsbaum (T, δ) für IG_k mit $boolw(T, \delta) = \log_2\left(\frac{k\cdot(k-1)}{2} + 1\right)$. Es folgt also

$$\begin{aligned} boolw(IG_k) &\leq \log_2\left(\frac{k\cdot(k-1)}{2} + 1\right) \;=\; \log_2\left(\frac{1}{2}\cdot\left(k^2 - k + 2\right)\right) \\ &\leq \log_2\left(\frac{k^2}{2}\right) \;=\; \log_2(k^2) - \log_2(2) \;=\; 2\cdot\log_2(k) - 1 \end{aligned}$$

und damit die Behauptung. □

Durch Kombinieren von Folgerung 4.5.3 und Lemma 4.5.4 folgt sofort:

Hauptsatz 4.5.5 *Für die Boolsche Weite der I-Gitter IG_k mit $k \geq 2$ gilt:*

$$\log_2(k) - 1 \leq boolw(IG_k) \leq 2\cdot\log_2(k) - 1$$

Auch hier liegt, aufgrund der Kantenstruktur des Graphen IG_k, die Vermutung nahe, dass die obere Schranke näher an der tatsächlichen Boolschen Weite liegt, als es die untere tut. Dazu müsste auch hier die untere Schranke verbessert werden, ein mögliches Werkzeug wäre, wie auch bei einer möglichen Verbesserung der unteren Schranke an die Boolsche Weite der H-Gitter, der Satz 4.3.1.

4.6 Sichere Separatoren

Zum Abschluss dieses Kapitels soll nun noch einmal Rückbezug auf die vorherigen Untersuchungen genommen werden, in wie weit Aussagen für Baumweite Gültigkeit haben für die Boolsche Weite. Denn für Baumweite gibt es noch ein weiteres interessantes Konzept: die sogenannten sicheren Separatoren (engl. *safe separators*).

Definition 4.6.1 *Es sei $G = (V, E)$ ein zusammenhängender Graph, $S \subseteq V$ ein Separator und $D_1, D_2, \ldots, D_d$ die Zusammenhangskomponenten von $G - S$. Der Separator ist genau dann **sicher** für Baumweite, wenn $tw(G) = \max\limits_{1\leq i\leq d} tw(G[V(D_i)\uplus S])$ gilt. Völlig analog werden sichere Separatoren für Boolsche Weite definiert.*

Sichere Separatoren sind ein sehr nützliches Werkzeug, wenn die Baumweite eines Graphen bestimmt werden soll. Ist ein sicherer Separator S in G bekannt, so kann die Analyse auf die Komponenten von $G - S$ und auf S beschränkt werden. Im Idealfall kann so, durch iteratives Bestimmen von sicheren Separatoren, das Problem, die Baumweite von G zu berechnen, stark vereinfacht werden.

Da Separatoren - und insbesondere inklusionsminimale - Separatoren über bekannte Wege effizient bestimmt werden können (siehe Takata [2010]), drängt sich daher die Frage auf, wann ein Separator als sicher zu klassifizieren ist. Hierzu ein sehr schönes Ergebnis.

Lemma 4.6.2 (Olesen und Madsen [1999]) *Es sei G ein Graph und $S \subseteq V(G)$ ein Separator. Ist S eine Clique, so ist S sicher für Baumweite*

Einige andere Klassifizierungen für sichere Separatoren und weitere sehr interessante Ergebnisse sind in Bodlaender und Koster [2003] zu finden. Ziel soll es aber nun sein zu untersuchen, ob Cliquenseparatoren ebenfalls sicher für Boolsche Weite sind oder nicht.

Entscheidendes Hilfsmittel hierzu werden die bereits in Abschnitt 4.5 vorgestellten I-Gitter sein, vgl. Definition 4.5.1. Nach Folgerung 4.5.3 gilt hier einerseits $boolw(IG_k) \geq \log_2(k) - 1$ für alle $k \in \mathbb{N}$. Als beste obere Schranke an die Boolsche Weite ergab sich mit Lemma 4.5.4 hingegen $boolw(IG_k) \leq \log_2\left(\frac{k\cdot(k-1)}{2} + 1\right)$, was vermutlich auch minimal ist, allerdings zu zeigen verbleibt. Unter Berücksichtigung dieser Umstände sind, im Kontrast zu Lemma 4.6.2, die beiden folgenden Aussagen zu sehen.

Lemma 4.6.3 *Es folgt $boolw(IG_k) \leq \log_2(k)$, wenn Cliquenseparatoren für Boolsche Weite sicher sind.*

Beweis
Nach Definition der I-Gitter bildet die Menge $S_\ell := \{v_{i,\ell} \in V(IG_k) : 1 \leq i \leq k\}$ für alle $1 \leq \ell \leq k$ eine k-Clique in IG_k. Sind Cliquenseparatoren also sicher für die Boolsche Weite, so folgt induktiv sofort:

$$boolw(IG_k) = \max_{1\leq\ell\leq k-1} boolw(IG_k[S_\ell \uplus S_{\ell+1}])$$

Der Graph $IG_k[S_\ell \uplus S_{\ell+1}]$ ist aber für alle $1 \leq \ell \leq k-1$ ein induzierter Teilgraph (bzw. isomorph zu einem solchen) des H-Gitters HG_k, wie mit Blick auf die Definition 3.4.21 unmittelbar folgt. Durch Kombination von Satz 3.4.17 und Lemma 4.4.1 gilt daher $boolw(IG_k[S_\ell \uplus S_{\ell+1}]) \leq boolw(HG_k) \leq \log_2(k)$ und die Behauptung folgt. □

Direkt aus Lemma 4.6.3 ergibt sich:

Folgerung 4.6.4 *Gilt die Vermutung $boolw(IG_k) > \log_2(k)$, so sind Cliquenseparatoren beliebiger Ordnung für die Boolsche Weite im Allgemeinen nicht sicher.*

Wird die beste bisher bekannte untere Schranke an die Boolsche Weite des I-Gitters nur minimal (genauer einen beliebigen Wert > 1) verbessert, so sind Cliquenseparatoren also nicht sicher für die Boolsche Weite eines Graphen. Da diese untere Schranke $boolw(IG_k) \geq \log_2(k) - 1$ alleine aus der Cliquenweite der I-Gitter folgt, ist es also entsprechend wahrscheinlich, dass Cliquenseparatoren nicht als sichere Separatoren für Boolsche Weite in Frage kommen.

Interessant ist allerdings, ob es grundsätzlich keine sicheren Separatoren für Boolsche Weite gibt. Bei der Analyse der Gittergraphen G_k ist bereits aufgefallen, dass ein Schnitt $(A, \overline{A})$, bei dem die Schnittkanten ein perfektes Machting induzieren, relativ ungünstig zur Minimierung von $|UN(A)|$ ist: Jedes $X \subseteq A$ hat eine Nachbarschaft $N(X) \cap \overline{A}$ und X ist, da die Schnittkanten ein perfektes Matching induzieren, durch diese Eigenschaft bereits eindeutig bestimmt. Es folgt unmittelbar $|UN(A)| = 2^{|A \cap N(\overline{A})|}$. Daher drängt sich die Frage auf, ob Matchingseparatoren sicher sind für die Boolsche Weite. Doch auch das trifft leider nicht zu.

Lemma 4.6.5 *Matchingseparatoren beliebiger Größe sind für $k \geq 6$ nicht sicher für die Boolsche Weite.*

Beweis

Es sei M_k ein Matching mit k Kanten und entsprechend $2 \cdot k$ Knoten. Betrachte dann den Gittergraphen G_k, vgl. Definition 3.4.14. Wird $Z_\ell := \{v_{\ell,j} \in V(G_k) : 1 \leq j \leq k\}$ definiert, so ist der Teilgraph $G_k[Z_{\lfloor \frac{k}{2} \rfloor} \uplus Z_{\lfloor \frac{k}{2} \rfloor + 1}] \cong M_k$.

Die beiden eindeutigen Komponenten von $G_k - \left(Z_{\lfloor \frac{k}{2} \rfloor} \uplus Z_{\lfloor \frac{k}{2} \rfloor + 1}\right)$ sind dann gerade die Teilgraphen $G_k\left[Z_1 \uplus \cdots \uplus Z_{\lfloor \frac{k}{2} \rfloor - 1}\right]$ sowie $G_k\left[Z_{\lfloor \frac{k}{2} \rfloor + 2} \uplus \cdots \uplus Z_k\right]$. Wären nun Matchingseparatoren sicher für die Boolsche Weite, so folgt mit Lemma 4.3.25:

$$\begin{aligned} boolw(G_k) &= \max\left\{boolw\left(G_k\left[Z_1 \uplus \cdots \uplus Z_{\lfloor \frac{k}{2} \rfloor + 1}\right]\right),\ boolw\left(G_k\left[Z_{\lfloor \frac{k}{2} \rfloor} \uplus \cdots \uplus Z_k\right]\right)\right\} \\ &= \max\{boolw\left(G_{(\lfloor \frac{k}{2} \rfloor + 1) \times k}\right),\ boolw\left(G_{(\lceil \frac{k}{2} \rceil + 1) \times k}\right)\} \\ &\leq boolw\left(G_{(\lceil \frac{k}{2} \rceil + 1) \times k}\right) \leq boolw\left(G_{\lceil \frac{k}{2} \rceil + 2}\right) \end{aligned}$$

Unter der Voraussetzung $k \geq 6$ ist aber $\lceil \frac{k}{2} \rceil + 2 < k$, also folgt aus Hauptsatz 4.3.22 und Lemma 4.3.21, dass $boolw(G_{\lceil \frac{k}{2} \rceil + 2}) < boolw(G_k)$ ist. Das ist ein Widerspruch, also war die Annahme falsch, dass Matchingseparatoren sicher für Boolsche Weite sind. □

Somit bleibt die Frage offen, ob es überhaupt sichere Separatoren für Boolsche Weite gibt. Die Ergebnisse dieses Abschnitts sind aber insofern hilfreich, als dass zwei vielversprechende Möglichkeiten ausgeschlossen bzw. vermutlich ausgeschlossen sind.

5 Lösbarkeit durch ganzzahlige lineare Programmierung

Ziel dieses Kapitels ist es, der Frage nachzugehen, in wie weit die Boolsche Weite eines Graphen bestimmt werden kann. Bisherige Arbeiten auf diesem Gebiet, wie etwa Hvidevold u. a. [2011] oder Bui-Xuan u. a., beschäftigen sich in erster Linie mit der Suche nach vollständigen Zerlegungsbäumen, deren Boolsche Weite möglichst klein ist und damit eine obere Schranke an die Boolsche Weite des Ausgangsgraphen liefern. Die Frage allerdings, ob diese Zerlegungsbäume auch optimal sind, wird leider nicht beantwortet.

Daher soll nun eine Möglichkeit vorgestellt werden, die es zumindest theoretisch möglich macht, diese Frage zu beantworten. Größtes und entscheidendes Hilfsmittel hierzu wird die ganzzahlige lineare Programmierung sein, da es durch geeignete Software dann möglich ist, eine optimale Lösung zu finden. Für eine ausführliche Einführung in dieses Gebiet sei an die Literatur, wie etwa Wolsey [1998], Schrijver [1986] oder Nemhauser und Wolsey [1988], verwiesen.

5.1 Konstruktion des ganzzahligen linearen Programms

Erstes Ziel ist es nun, ein ganzzahliges lineares Programm zu konstruieren, mit dem die Boolsche Weite eines Graphen bestimmt werden kann. Da dieses Problem direkt auf die Boolsche Weite eines vollständigen Zerlegungsbaumes führt, ist zunächst eine geeignete Repräsentation der Baumstruktur erforderlich.

Lemma 5.1.1 *Es sei $G = (V, E)$ ein Graph mit $|V| = n$ und (T, δ) ein vollständiger Zerlegungsbaum zu G. Dann hat T genau $2 \cdot n - 1$ viele Knoten, von denen n Blätter sind.*

Beweis
Es folgt sofort aus der Definition, dass jeder vollständige Zerlegungsbaum T genau n Blätter hat. Zeige also per vollständiger Induktion nach n, dass $|V(T)| = 2 \cdot n - 1$ gilt:

Für $n = 1$ ist die Behauptung klar, da sowohl G als auch T aus genau einem Knoten bestehen. Nun sei $H = (\{w_1, w_2, \ldots, w_{n+1}\}, E')$ ein Graph auf $n+1$ Knoten und (S, γ) ein vollständiger Zerlegungsbaum zu H. Es ist nach Konstruktion klar, dass S einen inneren Knoten i_0 hat mit $|\gamma(i_0)| = 2$. Dabei kann durch Umnummerieren der Knoten angenommen werden, dass $\gamma(i_0) = \{w_n, w_{n+1}\}$ gilt. Die Söhne von i_0 in S seien i_1 und i_2. Definiere (S', γ') nun wie folgt:

$$S' := S - \{i_1, i_2\} \text{ und für } i \in V(S') \text{ sei } \gamma'(i) := \gamma(i) \backslash \{w_{n+1}\}$$

(S', γ') geht also aus S hervor, indem w_{n+1} aus allen Mengen gelöscht wird, wodurch i_0 ein Blatt mit $\gamma'(i_0) = \{w_n\}$ wird. Somit ist (S', γ') ein vollständiger Zerlegungsbaum zu $H[\{w_1, \ldots, w_n\}]$, für den die Behauptung mit der Induktionsvoraussetzung zutrifft, also $|V(S')| = 2 \cdot n - 1$. Nach Konstruktion gilt $V(S) = V(S') \uplus \{i_1, i_2\}$, also ergibt sich:

$$|V(S)| = |V(S')| + 2 = 2 \cdot n - 1 + 2 = 2 \cdot (n+1) - 1$$

Es folgt die Behauptung. □

Ist (T, δ) nun ein vollständigen Zerlegungsbaumes zu $G = (V, E)$ mit $|V| = n$, kann mit Lemma 5.1.1 also o.B.d.A. angenommen werden, dass $V(T) = \{1, 2, \ldots, 2 \cdot n - 1\}$ ist. Damit kann die Baumstruktur durch binäre Variablen $x_{i,j}$ wie folgt repräsentiert werden:

$$x_{i,j} = \begin{cases} 1 & \text{falls i Vater von j in } T \text{ ist} \\ 0 & \text{sonst} \end{cases},$$

für alle $1 \leq i < j \leq 2 \cdot n - 1$. Die Forderung "$i < j$" ist grundsätzlich nicht notwendig, kann aber o.B.d.A. angenommen werden, da dies nur bedeutet, dass ein Knoten einen kleineren Index als seine Söhne haben muss. Insbesondere bedeutet dies, dass die Wurzel den Index 1 hat. Da jeder Knoten außer der Wurzel genau einen Vater und jeder Knoten entweder keine oder genau zwei Söhne hat, ergibt sich sofort:

Folgerung 5.1.2 *Für alle $2 \leq j \leq 2 \cdot n - 1$ gilt*

$$\sum_{i=1}^{j-1} x_{i,j} = 1 \tag{5.1}$$

und für alle $1 \leq i \leq 2 \cdot n - 1$ ist

$$\sum_{j=i+1}^{2n-1} x_{i,j} \in \{0, 2\}. \tag{5.2}$$

Um zu unterscheiden, ob $\sum_{j=i+1}^{2n-1} x_{i,j} = 0$ oder $= 2$ gilt, ist es wichtig zu wissen, ob i ein Blatt ist oder nicht. Durch eine weitere Einschränkung kann dies elegant gelöst werden.

Lemma 5.1.3 *Es sei (T, δ) ein vollständiger Zerlegungsbaum zu $G = (V, E)$ mit $|V| = n$ sowie $V(T) = \{1, \ldots, 2 \cdot n - 1\}$. Dann können die Knoten so benannt werden, dass die Menge der Blätter die Form $\mathcal{L}(T) = \{n, \ldots, 2 \cdot n - 1\}$ hat.*

Beweis

Die Aussage wird per Induktion nach $\ell := |\mathcal{L}(T)|$ gezeigt und ist für $\ell = 1$ trivial. Wie im Beweis von Lemma 5.1.1 sei wieder $H = (\{w_1, w_2, \ldots, w_{n+1}\}, E')$ ein Graph auf $n + 1$ Knoten und (S, γ) ein vollständiger Zerlegungsbaum zu H. Wähle auch hier einen inneren Knoten i_0 mit $\gamma(i_0) = \{w_n, w_{n+1}\}$, der nach geeigneter Umbenennung der Knoten existiert, und konstruiere (S', γ') zu $H[\{w_1, \ldots, w_n\}]$ ebenfalls auf gleiche Art. Auf S' kann jetzt die Induktionsvoraussetzung angewendet werden, wobei $V(S') = \{1, \ldots, 2 \cdot n - 1\}$ ist und i_0 Label n habe. Nach Lemma 5.1.3 kann $V(S) = \{1, \ldots, 2 \cdot (n+1) - 1\}$ angenommen werden, d.h. S hat zwei zusätzliche freie Labels (verglichen mit S'). Werden die beiden zusätzlichen Knoten in S jeweils mit den beiden zusätzlichen Zahlen gelabelt, so folgt die Behauptung. □

So kann die Aussage von Nebenbedingung (5.2) verschärft werden zu:

Folgerung 5.1.4 *Es kann o.B.d.A. angenommen werden, dass für alle $1 \leq i \leq n-1$*

$$\sum_{j=i+1}^{2n-1} x_{i,j} = 2 \tag{5.3}$$

und für alle $n \leq i \leq 2 \cdot n - 1$

$$\sum_{j=i+1}^{2n-1} x_{i,j} = 0 \tag{5.4}$$

gilt.

Damit ist die Modellierung für den Baum T abgeschlossen:

Satz 5.1.5 *Es sei $G = (V, E)$ ein Graph mit $|V(G)| = n$ und (T, δ) sei ein vollständiger Zerlegungsbaum zu G. Dann existiert ein zu T isomorphes $T' \cong T$, so dass $V(T') = \{1, \ldots, 2 \cdot n - 1\}$ und $\mathcal{L}(T') = \{n, \ldots, 2 \cdot n - 1\}$ gilt und mit*

$$x_{i,j} := \begin{cases} 1 & \text{falls } i < j \text{ und } \{i, j\} \in E(T') \\ 0 & \text{sonst} \end{cases},$$

die Gleichungen (5.1), (5.2), (5.3) und (5.4) erfüllt sind, d.h. es gilt

- *für alle* $2 \leq j \leq 2 \cdot n - 1$*:*
$$\sum_{i=1}^{j-1} x_{i,j} = 1$$
- *für alle* $1 \leq i \leq n - 1$*:*
$$\sum_{j=i+1}^{2n-1} x_{i,j} = 2$$
- *für alle* $n \leq i \leq 2 \cdot n - 1$*:*
$$\sum_{j=i+1}^{2n-1} x_{i,j} = 0$$

Beweis
Die Existenz eines solchen T' folgt aus den Lemmata 5.1.1 und 5.1.3, es bleibt zu zeigen, dass T' auch so gewählt werden kann, dass zusätzlich die Gleichungen erfüllt sind.

Betrachte dazu, für festes $\mathsf{j} \in V(T')$, die Nachbarschaft $N(\mathsf{j})$. Ebenfalls mit Lemma 5.1.3 folgt, dass die Wurzel Knoten 1 entspricht und, dass jedes $\mathsf{j} > 1$ ein eindeutiges $\mathsf{i} \in N(\mathsf{j})$ hat mit $\mathsf{i} < \mathsf{j}$. Das ist die Behauptung für (5.1). Da T' ein vollständiger Binärbaum ist, gilt

$$\forall v \in V(T') : d_{T'}(v) \in \{1, 2, 3\}$$

wobei die Wurzel Knoten 1 entspricht und eindeutig mit der Eigenschaft $d_{T'}(v) = 2$ ist. Da (5.1) gilt, folgt also für jedes $1 \leq \mathsf{i} \leq 2 \cdot n - 1$:

$$|N(\mathsf{i}) \cap \{\mathsf{j} \in \mathbb{N} : \mathsf{i} < \mathsf{j}\}| \in \{0, 2\}$$

Das ist (5.2) und mit Lemma 5.1.3 folgt daraus sofort (5.3) sowie (5.4). □

Weiter ist klar, dass jede Variablenbelegung, die (5.1), (5.2), (5.3) und (5.4) erfüllt, auf gleiche Art einen vollständigen Binärbaum definiert. Es gilt also nun aus dem Binärbaum, mit Hilfe einer Abbildung $\delta : \{1, \ldots, 2 \cdot n - 1\} \to Pot(V(G)) \backslash \{\emptyset\}$, einen (vollständigen) Zerlegungsbaum zu konstruieren, wobei G ein gegebener Graph ist. Dazu ein wichtiges Lemma.

Lemma 5.1.6 *Für jeden Graphen G und für jeden vollständigen Zerlegungsbaum $(T, \delta) \in \mathcal{T}_f(G)$ induziert δ ein Bijektion zwischen den Blättern $\mathcal{L}(T)$ von T und den Knoten $V(G)$ von G*

$$\delta^* : \mathcal{L}(T) \to V(G), \ell \mapsto \iota(\delta(\ell)),$$

wobei $\iota(\{v\}) := v$ für v in $V(G)$ sei. Weiter ist damit jeder vollständige Zerlegungsbaum über seine Baumstruktur und die δ-Labels seiner Blätter bereits eindeutig bestimmt.

Beweis
Nach Folgerung 3.3.2 bildet

$$\{\delta(\ell) : \ell \in \mathcal{L}(T)\}$$

eine Partition von V, d.h. δ^* ist die gewünschte Bijektion. Nun gilt nach Definition, dass ein Knoten $v \in V(T)\backslash\mathcal{L}(T)$ mit Söhnen s_1 und s_2 erfüllt

$$\delta(v) = \delta(s_1) \uplus \delta(s_2),$$

wobei alle Mengen nicht-leer sind. Also definieren zwei Blätter mit gleichen Vater bereits eindeutig dessen δ-Label u.s.w., es lassen sich also - wie eine triviale Induktion zeigt - von den Blättern bis zur Wurzel alle Mengen konstruieren. Das ist die Behauptung. □

Um dieses in einem linearen Programm umzusetzen, seien für alle $\mathsf{v} \in V(G)$ und für alle $n \leq \mathsf{i} \leq 2 \cdot n - 1$ binäre Variablen $\mathsf{y}_{\mathsf{v},\mathsf{i}}$ sowie für alle $1 \leq \mathsf{i} \leq 2 \cdot n - 1$ binäre Variablen $\mathsf{z}_{\mathsf{v},\mathsf{i}}$ wie folgt definiert:

$$\mathsf{y}_{\mathsf{v},\mathsf{i}} = \begin{cases} 1 & \text{falls } \delta(\mathsf{i}) = \{\mathsf{v}\} \text{ (d.h. } \mathsf{i} \text{ ist Blatt zu } \mathsf{v} \in V(G)) \\ 0 & \text{sonst} \end{cases}$$

$$\mathsf{z}_{\mathsf{v},\mathsf{i}} = \begin{cases} 1 & \text{falls } \mathsf{v} \in \delta(\mathsf{i}) \\ 0 & \text{falls } \mathsf{v} \notin \delta(\mathsf{i}) \end{cases}$$

Direkt aus der Konstruktion folgt:

Bemerkung 5.1.7 *Für alle $\mathsf{v} \in V(G)$ und für alle $n \leq i \leq 2 \cdot n - 1$ gilt:*

$$y_{v,i} = z_{v,i} \tag{5.5}$$

Weiter impliziert die Bijektion aus Lemma 5.1.6 sofort:

Folgerung 5.1.8 *Eine zulässige Lösung muss für $n \leq i \leq 2 \cdot n - 1$*

$$\sum_{v \in V(G)} y_{v,i} = 1 \tag{5.6}$$

und für alle $v \in V(G)$

$$\sum_{i=1}^{2n-1} y_{v,i} = 1 \tag{5.7}$$

erfüllen.

In diesem Sinne verankert die Bijektion δ^* die Labels aller Knoten über die Labels der Blätter des Zerlegungsbaumes (T, δ). Für eine weitere Beschreibung dieser Datenstruktur sind also nur noch Nebenbedingungen herzuleiten, so dass einerseits ein Knoten von T, der nicht die Wurzel ist, sein Label an seinen Vater (und somit an alle Vorgänger bis zur Wurzel) weitergibt und andererseits muss ein Knoten $t \in T$, der kein Blatt ist, jedes $\mathsf{v} \in \delta(t)$ an genau einen seiner beiden Söhne vererben.

Lemma 5.1.9 *Für alle* $n \leq i \leq 2 \cdot n - 1$ *und für alle* $v, w \in V(G)$ *mit* $v \neq w$ *gilt:*

$$y_{v,i} + z_{w,i} \leq 1 \tag{5.8}$$

Beweis
Die Ungleichung $\mathsf{y_{v,i} + z_{w,i} \leq 1}$ ist äquivalent zu $\mathsf{z_{w,i} \leq 1 - y_{v,i}}$. Ist also $\mathsf{y_{v,i} = 0}$, so ist sie trivialerweise erfüllt. Ist $\mathsf{y_{v,i} = 1}$, so gilt nach Definition $\delta(\mathsf{i}) = \{\mathsf{v}\}$. Dann folgt $\mathsf{w} \notin \delta(\mathsf{i})$ für alle $\mathsf{w} \neq \mathsf{v}$, also $\mathsf{z_{w,i} = 0}$. Es folgt die Behauptung. □

Die Ungleichung (5.8) besagt also, wenn i ein Blatt von T ist, so folgt $\delta(\mathsf{i}) = \{\mathsf{v}\}$ für ein entsprechendes $\mathsf{v} \in V(G)$. Die nun folgende Nebenbedingung beschäftigt sich mit folgernder Situation: Sind $\mathsf{i}, \mathsf{j} \in T$ und ist j Sohn von i mit $\mathsf{v} \in \delta(\mathsf{j})$, so muss nach Definition eines Zerlegungsbaumes auch $\mathsf{v} \in \delta(\mathsf{i})$ sein. Dies führt zur Nebenbedingung des nächsten Lemmas.

Lemma 5.1.10 *Für alle* $1 \leq i < j \leq 2 \cdot n - 1$ *und für alle* $v \in V(G)$ *gilt:*

$$x_{i,j} + z_{v,j} - z_{v,i} \leq 1 \tag{5.9}$$

Beweis
Die Ungleichung $\mathsf{x_{i,j} + z_{v,j} - z_{v,i} \leq 1}$ ist äquivalent zu $\mathsf{z_{v,i} \geq x_{i,j} + z_{v,j} - 1}$. Ist dann $\mathsf{x_{i,j} = 0}$ oder $\mathsf{z_{v,j} = 0}$, so ist die Bedingung trivial. Gilt hingegen $\mathsf{x_{i,j} = z_{v,j} = 1}$, so ist $\mathsf{v} \in \delta(\mathsf{j})$, da $\mathsf{z_{v,j} = 1}$, und i Vater von j, weil $\mathsf{x_{i,j} = 1}$ ist. Also muss nach Definition auch $\mathsf{v} \in \delta(\mathsf{i})$ gelten, was $\mathsf{z_{v,i} = 1}$ impliziert. Das ist die Behauptung. □

Es folgt nun noch eine Nebenbedingung, die im Prinzip analog zu Bedingung (5.9) zu sehen ist. Nur geht es dort um die Situation, dass sich ein Label vom Sohn j auf den Vater i fortsetzt. Andererseits muss sich aber auch für jeden Knoten i, der

kein Blatt ist, jedes $\mathsf{v} \in \delta(\mathsf{i})$ auf einen seiner beiden Söhne j oder k vererben. Dies formalisiert die nun folgende Aussage.

Lemma 5.1.11 *Für alle $v \in V(G)$ und für alle $1 \leq i < j, k \leq 2 \cdot n - 1$ mit $j \neq k$ gilt:*

$$x_{i,j} + x_{i,k} + z_{v,i} - z_{v,j} - z_{v,k} \leq 2 \tag{5.10}$$

Beweis
Hier liefert äquivalentes Umformen die Ungleichung $\mathsf{z_{v,j}} + \mathsf{z_{v,k}} \geq \mathsf{x_{i,j}} + \mathsf{x_{i,k}} + \mathsf{z_{v,i}} - 2$. Ist $\mathsf{x_{i,j}} = 0$, $\mathsf{x_{i,k}} = 0$ oder $\mathsf{z_{v,i}} = 0$, so ist die Bedingung an $\mathsf{z_{v,j}}$ und $\mathsf{z_{v,k}}$ also trivial. Andernfalls gilt $\mathsf{x_{i,j}} = \mathsf{x_{i,k}} = \mathsf{z_{v,i}} = 1$ und somit ist i sowohl Vater von j als auch von k, da $\mathsf{x_{i,j}} = \mathsf{x_{i,k}} = 1$ und $\mathsf{v} \in \delta(\mathsf{i})$, weil $\mathsf{z_{v,i}} = 1$ gilt. Dann folgt entweder $\mathsf{v} \in \delta(\mathsf{j})$ oder $\mathsf{v} \in \delta(\mathsf{k})$, was $\mathsf{z_{v,j}} = 1$ oder $\mathsf{z_{v,k}} = 1$ impliziert. Zusammen heißt das, dass $\mathsf{z_{v,j}} + \mathsf{z_{v,k}} \geq 1$ gilt. Das ist die Behauptung. □

Mit Hilfe dieser Nebenbedingungen ist ein vollständiger Zerlegungsbaum bereits eindeutig beschrieben, wie nun gezeigt wird.

Satz 5.1.12 *Es sei $G = (V, E)$ ein Graph mit $|V| = n$ Knoten und (T, δ) ein beliebiger vollständiger Zerlegungsbaum zu G. Dann induziert (T, δ) eine Variablenbelegung für $x_{i,j}$, $y_{v,i}$ und $z_{v,i}$ wie oben, die alle Nebenbedingungen (5.1) bis (5.10) erfüllt.*

Beweis
Die Existenz einer zulässigen Variablenbelegung für die $\mathsf{x_{i,j}}$ folgt aus Satz 5.1.5. Ferner stehen die Blätter $\mathcal{L}(T)$ nach Lemma 5.1.6 in Bijektion zu den Knoten von $V = V(G)$, so dass durch die Blätter sofort eine zulässige Belegung der $\mathsf{y_{v,i}}$ induziert wird. Also ist durch Nebenbedingung (5.5) auch die Belegung von $\mathsf{y_{v,i}}$ und $\mathsf{z_{v,i}}$ für alle $n \leq \mathsf{i} \leq 2 \cdot n - 1$ eindeutig festgelegt, die die Gleichungen (5.6) und (5.7) erfüllt. Zu zeigen bleibt also, dass auch die Ungleichungen (5.8), (5.9) und (5.10) stets gelten. Dies folgt aber sofort aus den Beweisen der Lemmata 5.1.9, 5.1.10 sowie 5.1.11. Es folgt die Behauptung. □

Mit Satz 5.1.12 ist also gewährleistet, dass es für jeden vollständigen Zerlegungsbaum eine entsprechende Variablenbelegung gibt, die alle hergeleiteten Nebenbedingungen erfüllt. Es soll daher nun gezeigt werden, dass eine zulässige Variablenbelegung ebenso einen zulässigen vollständigen Zerlegungsbaum induziert. Zu beachten ist, dass beide Sätze in sofern allgemein sind, dass sie lediglich von der Knotenanzahl n und in keinster Weise von der genauen Struktur, d.h. der Kantenmenge $E(G)$, abhängen.

Satz 5.1.13 *Es sei G ein Graph mit $|V(G)| = n$ Knoten. Weiter sei eine beliebige Belegung der Variablen $x_{i,j}$, $y_{v,i}$ und $z_{v,i}$ gegeben, die die Nebenbedingungen (5.1) bis (5.10) erfüllt. Dann induziert diese einen zulässigen vollständigen Zerlegungsbaum (T, δ) zu G.*

Beweis
Definiere $V(T) := \{1, \dots, 2 \cdot n - 1\}$ mit $\{i, j\} \in E(T)$ genau dann, wenn $x_{i,j} = 1$. Die Nebenbedingungen (5.1) bis (5.4) implizieren sofort, dass T dann ein vollständiger Binärbaum ist. Wird nun

$$\delta(i) := \{v \in V(G) : z_{v,i} = 1\}$$

für alle $i = 1, \dots, 2 \cdot n - 1$ definiert, so bleibt nach Definition 3.3.1 zu zeigen:

1. Für alle $i \in T$ ist $\delta(i) \neq \emptyset$.
2. Ist $r = 1$ die Wurzel von T, so folgt $\delta(r) = V$.
3. Ist $i \in T$ kein Blatt, so gilt für die Söhne j und k von i: $\delta(i) = \delta(j) \uplus \delta(k)$

Durch die Bedingungen (5.5) bis (5.8) steht $\mathcal{L}(T) = \{n, \dots, 2 \cdot n - 1\}$ in Bijektion zu $V(G)$, also gilt $\delta(i) \neq \emptyset$ für alle $i = n, \dots, 2 \cdot n - 1$, so dass Punkt 1.) nur noch für $i = 1, \dots, n - 1$ zu zeigen bleibt. Da die Ungleichung (5.9) für $1 \leq i < j \leq 2 \cdot n - 1$ und für alle $v \in V(G)$ erfüllt ist, folgt per trivialer Induktion, dass $v \in \delta(j)$ stets $v \in \delta(i)$ impliziert, wenn i Vorfahre von j ist. Da jedes $i = 1, \dots, n-1$ Vorfahre eines $j \in \mathcal{L}(T)$ ist, folgt also $\delta(i) \neq \emptyset$ auch für alle $i = 1, \dots, n - 1$.

Analog ist die Wurzel $r = 1$ Vorfahre eines jeden Blatts, so dass sofort $\delta(1) = V(G)$ folgt und somit nur noch Punkt 3.) zu zeigen ist. Dazu sei $i \in \{1, \dots, n - 1\}$ ein beliebiger Knoten, der kein Blatt ist, mit Söhnen j und k. Nach Definition ist dann $\delta(i) = \{v \in V(G) : z_{v,i} = 1\}$ (sowie analog $\delta(j)$ und $\delta(k)$), also ist zu zeigen, dass aus $z_{v,i} = 1$ entweder $z_{v,j} = 1$ oder $z_{v,k} = 1$ folgt, aber nicht beides gleichzeitig. Ersteres folgt sofort aus Ungleichung (5.10) und dem Beweis aus Lemma 5.1.11. Der zweite Teil wird indirekt gezeigt. Daher sei angenommen, dass für ein beliebiges 4-Tupel (v, i, j, k) die Belegung

$$x_{i,j} = x_{i,k} = z_{v,i} = z_{v,j} = z_{v,k} = 1$$

gelte. Anschaulich bedeutet dies, dass zwar $\delta(i) = \delta(j) \cup \delta(k)$ erfüllt, aber durch die zusätzliche Bedingung $z_{v,j} = 1 = z_{v,k}$, also $v \in \delta(j) \cap \delta(k)$, die Disjunktheit der beiden Mengen verletzt ist. Nach Definition müssen nun j und k weiter zerlegt werden, bis letztlich die Blätter genau einen Knoten aus G enthalten.

Damit existieren ℓ_1 und ℓ_2 mit $\delta(\ell_1) = \delta(\ell_2) = \{\mathsf{v}\}$, d.h. $\mathsf{y}_{\mathsf{v},\ell_1} = \mathsf{y}_{\mathsf{v},\ell_2} = 1$ gilt. Daraus folgt $\sum_{i=1}^{2n-1} \mathsf{y}_{\mathsf{v},\mathsf{i}} = 2$, also ist Nebenbedingung (5.7) verletzt. Das ist ein Widerspruch, also gilt auch die Disjunktheit in Punkt 3.) und insgesamt folgt die Behauptung. □

Die letzten beiden Sätze sagen zusammen aus, dass die Modellierung eines vollständigen Zerlegungsbaumes mit den Variablen $\mathsf{x}_{\mathsf{i},\mathsf{j}}$, $\mathsf{y}_{\mathsf{v},\mathsf{i}}$ sowie $\mathsf{z}_{\mathsf{v},\mathsf{i}}$, und den Nebenbedingungen (5.1) bis (5.10), bereits vollständig möglich ist und die beiden Modelle problemlos ineinander überführt werden können. Um das eigentliche Problem, nämlich einen optimalen vollständigen Zerlegungsbaum für einen gegebenen Graphen G zu bestimmen, mit Hilfe der linearen Optimierung zu lösen, bedarf es nun einer geeigneten Zielfunktion. Diese wird, als einziger Baustein des linearen Programms, natürlich von der Größe $|UN(\delta(\mathsf{i}))|$ - und damit unmittelbar von der Struktur von G selbst - abhängen. Nach Definition ist

$$boolw(G) = \min_{(T,\delta)\in\mathcal{T}_f(G)} boolw(T,\delta) = \min_{(T,\delta)\in\mathcal{T}_f(G)} \max_{t\in T} \log_2 |UN(\delta(t))|,$$

wobei $\mathcal{T}_f(G)$ die Menge aller vollständigen Zerlegungsbäume zu G bezeichnet, vgl. Definition 3.3.1. Da der Logarithmus injektiv ist, lässt sich dies etwas vereinfachen.

Bemerkung 5.1.14 *Für jeden Graphen G gilt:*

$$boolw(G) = \log_2 \left(\min_{(T,\delta)\in\mathcal{T}_f(G)} \max_{t\in T} |UN(\delta(t))| \right) \tag{5.11}$$

Listing 5.1: Algorithmus `UN`

```
1  UN(A) {
2     /* Zu A ⊆ V(G) wird |UN(A)| berechnet */
3
4     S1 := { v ∈ A | ∃ w ∈ Ā : {v,w} ∈ E };
5     S2 := { w ∈ Ā | ∃ v ∈ A : {v,w} ∈ E };
6
7     /* LN enthält später alle Nachbarschaften */
8     LN := { ∅ };
9
10    for u ∈ S1 do
11       for Y ∈ LN do
12          X := (N(u) ∩ S2) ∪ Y;
13          if (X ∉ LN)
14             LN := LN ∪ {X};
15
16    return |LN|;
17 }
```

Die Vereinfachung aus Bemerkung 5.1.14 ist für ein lineares Programm sehr nützlich, da der Ausdruck $\min_{(T,\delta)\in\mathcal{T}_f(G)} \max_{t\in T} |UN(\delta(t))|$ als mögliche lineare Zielfunktion wesentlich besser geeignet ist, als die ursprüngliche Definition mit Logarithmus. Um $|UN(\delta(t))|$, für ein gegebenes $\delta(t)$, bestimmen zu können, hilft der in Listing 5.1 abgebildete Algorithmus aus Hvidevold u. a. [2011]. Eine wichtige Aussage soll hier festgehalten werden.

Lemma 5.1.15 *Es sei $(A, \overline{A})$ ein Schnitt im Graphen G. Dann berechnet Algorithmus 5.1 den Wert $|UN(A)|$, er arbeitet also korrekt.*

Beweis
Im Algorithmus wird $(A, \overline{A})$ zunächst eingeschränkt auf (S_1, S_2) betrachtet, so dass $S_1 = A \cap N(\overline{A})$ sowie $S_2 = \overline{A} \cap N(A)$ ist. Dies ist keine Einschränkung, wie bereits in Folgerung 4.1.2 gesehen wurde. Es bleibt also zu zeigen, dass für jede Menge $M \subseteq A \cap N(\overline{A}) = S_1$ die Menge $N(M) \cap \overline{A}$ genau einmal in LN gespeichert wird. Durch die Abfrage in Zeile 13 wird keine Menge mehr als einmal gespeichert, daher muss nur gezeigt werden, dass jedes $N(M) \cap \overline{A}$ in LN vorkommt. Dies wird per vollständiger Induktion nach $k = |S_1|$ gezeigt.

Für $k = 0$, also $S_1 = \emptyset$, folgt $M = \emptyset$ und damit $N(M) \cap \overline{A} = \emptyset \in LN$ nach Definition von LN, siehe Zeile 8. Nun sei $|S_1| = k + 1 > 0$. Wird ein beliebiger Knoten $v \in S_1$ gewählt und $\widetilde{S}_1 := S_1 - \{v\}$ definiert, so wird für jedes $M \subseteq \widetilde{S}_1$ die Menge $N(M) \cap \overline{A}$ bereits nach Induktionsvoraussetzung in LN gespeichert. Wähle daher nun ein beliebiges $M \subseteq S_1$. Ist $v \notin M$ folgt $N(M) \cap \overline{A} \in LN$, es gelte also $v \in M$. Betrachte in diesem Fall $u = v$ in Zeile 10 sowie $Y = (N(M - \{v\})) \cap S_2 \in LN$ in Zeile 11. Dann gilt:

$$\begin{aligned} N(M) \cap \overline{A} &= \Big(N(v) \cap \overline{A}\Big) \cup \Big(N(M - \{v\}) \cap \overline{A}\Big) \\ &= (N(u) \cap S_2) \cup (N(M - \{v\}) \cap S_2) \\ &= (N(u) \cap S_2) \cup Y \end{aligned}$$

Die Anweisung aus Zeile 13 stellt damit sicher, dass $N(M) \cap \overline{A}$ in LN gespeichert wird, falls dies nicht bereits durch eine andere Menge getan wurde. Es folgt also die Behauptung auch für $|S_1| = k+1$, dies beendet die Induktion, so dass insgesamt die Behauptung folgt. □

Mit Hilfe von Algorithmus 5.1 und Lemma 5.1.15 kann also angenommen werden, dass der Wert $UN(A)$ für alle $A \subseteq V(G)$ stets bekannt ist. Es ist natürlich für größere Graphen absolut unpraktikabel, alle diese $2^{|V(G)|}$ vielen Mengen konkret zu bestimmen und jeweils den Wert $|UN(A)|$ einzeln auszurechnen.

Dennoch soll dieser Umstand für die Konstruktion des linearen Programms hier erst einmal vernachlässigt werden.

Satz 5.1.16 *Es sei (T, δ) ein vollständiger Zerlegungsbaum des Graphen G. Für alle $i \in T$ und alle $v \in V(G)$ sei Wert $z_{v,i}$ wie oben, also:*

$$z_{v,i} = \begin{cases} 1 & \text{falls } v \in \delta(i) \\ 0 & \text{falls } v \notin \delta(i) \end{cases}$$

Dann gilt

$$\max_{t \in T} |UN(\delta(t))| = \max_{A \subseteq V(G)} |UN(A)| \cdot \left(\sum_{v \in A} z_{v,i} + \sum_{v \in \overline{A}} (1 - z_{v,i}) - (|V(G)| - 1) \right)$$

für alle $A \subseteq V(G)$ und für alle $i \in T$.

Beweis
Zunächst wird folgende Aussage gezeigt: Ist $A = \delta(\mathsf{i})$, für ein beliebiges $\mathsf{i} \in T$, so folgt

$$\sum_{\mathsf{v} \in A} \mathsf{z}_{\mathsf{v},\mathsf{i}} + \sum_{\mathsf{v} \in \overline{A}} (1 - \mathsf{z}_{\mathsf{v},\mathsf{i}}) - (|V(G)| - 1) = 1$$

und andernfalls ist

$$\sum_{\mathsf{v} \in A} \mathsf{z}_{\mathsf{v},\mathsf{i}} + \sum_{\mathsf{v} \in \overline{A}} (1 - \mathsf{z}_{\mathsf{v},\mathsf{i}}) - (|V(G)| - 1) \leq 0.$$

Hierbei gilt:

$$\begin{aligned} & \sum_{\mathsf{v} \in A} \mathsf{z}_{\mathsf{v},\mathsf{i}} + \sum_{\mathsf{v} \in \overline{A}} (1 - \mathsf{z}_{\mathsf{v},\mathsf{i}}) - (|V(G)| - 1) = 1 \\ \iff & \sum_{\mathsf{v} \in A} \mathsf{z}_{\mathsf{v},\mathsf{i}} + \sum_{\mathsf{v} \in \overline{A}} (1 - \mathsf{z}_{\mathsf{v},\mathsf{i}}) = |V(G)| \\ \iff & \forall \mathsf{v} \in A : \mathsf{z}_{\mathsf{v},\mathsf{i}} = 1 \ \wedge \ \forall \mathsf{v} \in \overline{A} : \mathsf{z}_{\mathsf{v},\mathsf{i}} = 0 \\ \iff & A = \delta(\mathsf{i}) \end{aligned}$$

Dann folgt also:

$$\begin{aligned}
&\max_{t\in T} |UN(\delta(t))| = \max\{|UN(A)| : t \in T \ \wedge \ A = \delta(t)\} \\
&= \max\left\{|UN(A)| \cdot \left(\sum_{v\in A} z_{v,i} + \sum_{v\in\overline{A}} (1 - z_{v,i}) - (|V(G)| - 1)\right) : A = \delta(i)\right\} \\
&= \max\left\{|UN(A)| \cdot \left(\sum_{v\in A} z_{v,i} + \sum_{v\in\overline{A}} (1 - z_{v,i}) - (|V(G)| - 1)\right) : A \subseteq V(G)\right\} \\
&= \max_{A\subseteq V(G)} |UN(A)| \cdot \left(\sum_{v\in A} z_{v,i} + \sum_{v\in\overline{A}} (1 - z_{v,i}) - (|V(G)| - 1)\right)
\end{aligned}$$

Das ist die Behauptung. □

Zum praktischen Nutzen des Satzes 5.1.16 ist Folgendes zu sagen: Zunächst lässt sich die Aussage reformulieren zu

$$\max_{t\in T} |UN(\delta(t))| \geq |UN(A)| \cdot \left(\sum_{v\in A} z_{v,i} + \sum_{v\in\overline{A}} (1 - z_{v,i}) - (|V(G)| - 1)\right) \tag{5.12}$$

für alle $A \subseteq V(G)$ und die Ungleichung ist für mindestens eine Menge stets mit Gleichheit erfüllt. Die rechte Seite hängt dabei allerdings nur über den Index i vom jeweiligen Zerlegungsbaum (T, δ) ab. Nach Lemma 5.1.1 können die Knoten von T aber als $1, \ldots, 2 \cdot |V(G)| - 1$ angenommen werden, womit durch Satz 5.1.16 bzw. die Ungleichungen (5.12) der Term $\max_{t\in T} |UN(\delta(t))|$ durch lineare Nebenbedingungen ersetzt werden kann. Mit Blick auf Bemerkung 5.1.14 ergibt sich also eine wichtige Folgerung.

Folgerung 5.1.17 *Für einen festen Graphen G sei $boolw$ eine nicht-negative, ganzzahlige Variable, so dass*

$$boolw \geq |UN(A)| \cdot \left(\sum_{v\in A} z_{v,i} + \sum_{v\in\overline{A}} (1 - z_{v,i}) - (|V(G)| - 1)\right)$$

für alle $A \subseteq V(G)$ und für alle $i = 1, \ldots, 2 \cdot |V(G)| - 1$ gilt. Dann folgt:

$$boolw(G) \geq \log_2(boolw) \tag{5.13}$$

Minimieren der Variablen $boolw$ liefert also, bis auf Logarithmieren, die Boolsche Weite von G.

Insgesamt ergibt sich damit das abgebildete ganzzahlige lineare Programm 5.1.1 zur Bestimmung der Boolschen Weite $boolw(G)$, für einen beliebigen Graphen G mit $|V(G)| = n$.

Der Abschnitt soll mit folgendem Satz beendet werden, der die Korrektheit des ganzzahligen linearen Programms beschreibt, d.h. theoretisch für jeden Graphen G, mit Hilfe des linearen Programms, die Boolsche Weite $boolw(G)$ sowie ein optimaler vollständiger Zerlegungsbaum bestimmt werden kann. Der Beweis folgt sofort aus den Sätzen 5.1.12, 5.1.13 und 5.1.16 sowie Folgerung 5.1.17.

Hauptsatz 5.1.18 *Für jeden Graphen G und jeden optimalen vollständigen Zerlegungsbaum (T, δ) zu G gibt es eine zulässige Variablenbelegung für das ganzzahlige lineare Programm 5.1.1, so dass für den Zielfunktionswert* boolw *gilt:*

$$\log_2(\mathsf{boolw}) = boolw(T, \delta) = boolw(G)$$

In diesem Sinne ist die ganzzahlige lineare Optimierung also eine Möglichkeit, um einen optimalen vollständigen Zerlegungsbaum bzw. die Boolsche Weite eines gegebenen Graphens zu bestimmen. Allerdings ist dies teilweise noch relativ schlecht gelöst, so dass sich nun noch einer besseren Effizienz gewidmet wird.

5.1.1 Ganzzahliges lineares Programm zur Boolschen Weite

$$\min \ \mathsf{boolw}$$

$$\text{s.t.} \quad |UN(A)| \cdot \left(\sum_{\mathsf{v} \in A} \mathsf{z_{v,i}} + \sum_{\mathsf{v} \in \overline{A}} (1 - \mathsf{z_{v,i}}) - (n-1) \right) \leq \mathsf{boolw} \qquad \forall A \subseteq V(G),\ \forall 1 \leq \mathsf{i} \leq 2n-1$$

$$\sum_{\mathsf{i}=1}^{\mathsf{j}-1} \mathsf{x_{i,j}} = 1 \qquad \forall 2 \leq \mathsf{j} \leq 2n-1$$

$$\sum_{\mathsf{j}=\mathsf{i}+1}^{2n-1} \mathsf{x_{i,j}} = 2 \qquad \forall 1 \leq \mathsf{i} \leq n-1$$

$$\sum_{\mathsf{j}=\mathsf{i}+1}^{2n-1} \mathsf{x_{i,j}} = 0 \qquad \forall n \leq \mathsf{i} \leq 2n-1$$

$$\sum_{\mathsf{v} \in V(G)} \mathsf{y_{v,i}} = 1 \qquad \forall n \leq \mathsf{i} \leq 2n-1$$

$$\sum_{\mathsf{i}=1}^{2n-1} \mathsf{y_{v,i}} = 1 \qquad \forall \mathsf{v} \in V(G)$$

$$\mathsf{y_{v,i}} - \mathsf{z_{v,i}} = 0 \qquad \forall \mathsf{v} \in V(G),\ \forall n \leq \mathsf{i} \leq 2n-1$$

$$\mathsf{y_{v,i}} + \mathsf{z_{w,i}} \leq 1 \qquad \forall \mathsf{v} \neq \mathsf{w} \in V(G),\ \forall n \leq \mathsf{i} \leq 2n-1$$

$$\mathsf{x_{i,j}} + \mathsf{z_{v,j}} - \mathsf{z_{v,i}} \leq 1 \qquad \forall \mathsf{v} \in V(G),\ \forall 1 \leq \mathsf{i} < \mathsf{j} \leq 2n-1$$

$$\mathsf{x_{i,j}} + \mathsf{x_{i,k}} + \mathsf{z_{v,i}} - \mathsf{z_{v,j}} - \mathsf{z_{v,k}} \leq 2 \qquad \forall \mathsf{v} \in V(G),\ \forall \mathsf{i} < \mathsf{j}, \mathsf{k},\ \mathsf{j} \neq \mathsf{k}$$

$$\mathsf{x_{i,j}} \in \{0,1\} \qquad \forall 1 \leq \mathsf{i}, \mathsf{j} \leq 2n-1$$

$$\mathsf{y_{v,i}} \in \{0,1\} \qquad \forall \mathsf{v} \in V(G),\ \forall 1 \leq \mathsf{i} \leq 2n-1$$

$$\mathsf{z_{v,i}} \in \{0,1\} \qquad \forall \mathsf{v} \in V(G),\ \forall 1 \leq \mathsf{i} \leq 2n-1$$

$$\mathsf{boolw} \in \mathbb{N}$$

Ganzzahliges lineares Programm für die Boolsche Weite eines Graphen G

5.2 Vermeidung von Symmetrien

Das ganzzahlige lineare Programm 5.1.1 zur Bestimmung der Boolsche Weite eines Graphen G stellt nach dem Hauptsatz 5.1.18 grundsätzlich eine Möglichkeit dar, die Boolsche Weite eines Graphen sowie einen vollständigen Zerlegungsbaum zu bestimmen. Eine große Schwäche besteht allerdings noch darin, dass sehr viele Variablenbelegungen tatsächlich den selben vollständigen Zerlegungsbaum induzieren. Naheliegende Gründe dafür sind einerseits, dass jede beliebige Bijektion zwischen den Blättern des Zerlegungsbaumes und den Knoten aus $V(G)$ grundsätzlich möglich ist. Damit kann die Belegung der $\mathsf{y}_{\mathsf{v,i}}$ Variablen beliebig gewählt werden, insofern sie tatsächlich eine Bijektion definiert, also zulässig ist. Der induzierte vollständige Zerlegungsbaum ändert sich aber nicht, insofern die verbleibenden Variablen entsprechend angepasst werden. Anderseits können die Labels der Knoten des vollständigen Zerlegungsbaumes auch weitestgehend beliebig gewählt werden, die einzige Bedingung ist bisher, dass die Blätter mit den Labels n bis $2 \cdot n - 1$ versehen werden, und ein innerer Knoten t immer ein kleineres Label als seine Söhne (und damit induktiv als alle seine Nachfahren) haben muss. Beide Problematiken sollen mit einem kurzen Beispiel veranschaulicht werden.

Beispiel 5.2.1 Mit $G = (V, E) \cong P_5$ sei ein Pfad auf fünf Knoten gegeben, d.h. $V = \{1, 2, 3, 4, 5\}$ und $E = \{\{i, i+1\} : 1 \leq i \leq 4\}$. Ein optimaler vollständiger Zerlegungsbaum mit Boolscher Weite $\log_2(2) = 1$ lässt sich wie folgt konstruieren:

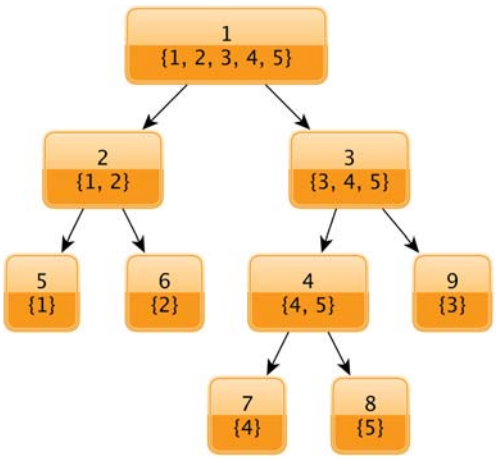

Abbildung 5.1: Ein optimaler vollständiger Zerlegungsbaum zu einem P_5

Die einzelne Ziffer der Knotenbeschriftung stellt den Knotenindex im Sinne des ganzzahligen linearen Programms dar und die Menge darunter beschreibt das δ-Label.

Hier ist zu beachten, dass der eigentliche Zerlegungsbaum keine Knotenindizes wie im ganzzahligen linearen Programms hat, daher sind zwei Lösungen des ganzzahligen linearen Programms, die sich nur in der Indizierung der Knoten unterscheiden, als gleich anzusehen. Es sei daher mit $(\mathsf{i}, \mathsf{j}, \mathsf{k}, \mathsf{l}, \mathsf{m})$ eine beliebige Permutation der Menge der Blätter des Zerlegungsbaumes $\mathcal{L}(T) = \{5, 6, 7, 8, 9\}$ gegeben. Für jede dieser $5! = 120$ Permutationen ist der Zerlegungsbaum aus Abbildung 5.2 eine Visualisierung einer zulässigen Lösung, die rein formal unterschiedlich sind, tatsächlich aber den gleichen Zerlegungsbaum beschreiben.

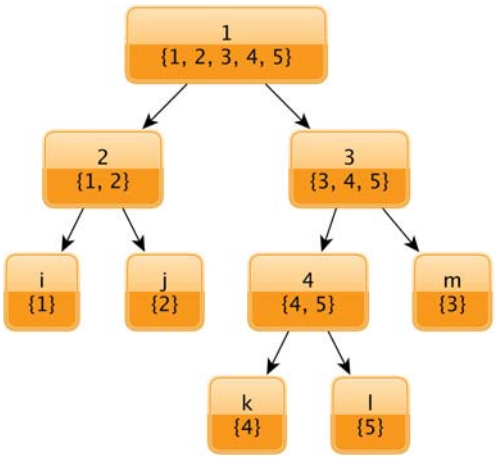

Abbildung 5.2: Eine alternative Familie $5!$ vollständiger Zerlegungsbäume zum gleichen Graphen P_5

In einer etwas formaleren Herangehensweise ließe sich zeigen, dass die symmetrische Gruppe auf $|V(G)| = n$ Punkten S_n auf der Menge aller Lösungen des ganzzahligen linearen Programms operiert, tatsächlich aber ein Vertreter pro Bahn dieser Operation ausreicht, um alle Lösungen erfassen zu können. Bevor sich einer Lösung dieses Problems gewidmet wird, soll aber kurz noch ein ähnliches, wenn auch nicht ganz so ausuferndes, Symmetrieproblem vorgestellt werden.

Beispiel 5.2.2 Betrachte erneut den Graphen P_5 wie in Beispiel 5.2.1 sowie den vollständigen Zerlegungsbaum aus Abbildung 5.1. Nun sei $(\mathsf{i}, \mathsf{j}, \mathsf{k})$ eine beliebige Permutationen der drei inneren Knoten $\{2, 3, 4\}$, so dass $\mathsf{j} < \mathsf{k}$ gilt, womit die Möglichkeiten $(\mathsf{i}, \mathsf{j}, \mathsf{k}) \in \{(2, 3, 4), (3, 2, 4), (4, 2, 3)\}$ verbleiben. So ergeben sich drei weitere unterschiedliche Lösungen des ganzzahligen linearen Programms aus Abbildung 5.3, die tatsächlich aber alle den gleichen vollständigen Zerlegungsbaum beschreiben.

Festzuhalten ist weiterhin, dass die Problematiken aus den Beispielen 5.2.1 und 5.2.2 natürlich auch simultan auftreten. So ergeben sich selbst für dieses kleine Beispiel

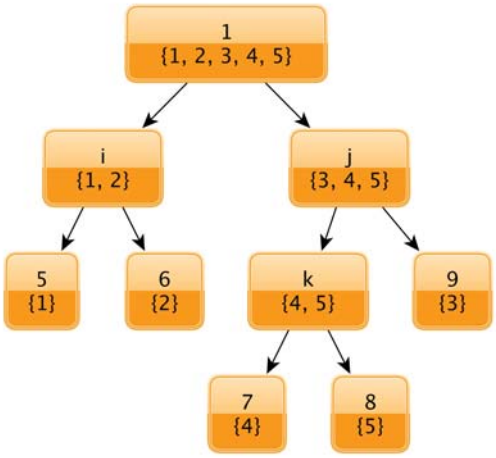

Abbildung 5.3: Eine Möglichkeit drei weitere vollständiger Zerlegungsbäume zum gleichen Graphen P_5 zu konstruieren

eines Graphens mit fünf Knoten, dass es $5! \cdot 3 = 360$ äquivalente Beschreibungen für den gleichen vollständigen Zerlegungsbaum gibt.

Entsprechend hat jede zulässige Lösung des ganzzahligen linearen Programms, insofern nur die bisherigen Nebenbedingungen angenommen werden, in gleicher Art und Weise mindestens $\Omega(n!)$ äquivalente Beschreibungen. Dies ist offensichtlich selbst für kleine Graphen sehr schnell unpraktikabel groß. Auch wenn sich diese Symmetrien leider nicht in Gänze verhindern lassen, so soll es dennoch Ziel dieses Abschnitts sein, die Nebenbedingungen des ganzzahligen linearen Programms wenigstens möglichst gut zu verbessern, um so geringere Laufzeiten zu erzielen.

Begonnen werden soll mit einer Lösung des Problems aus Beispiel 5.2.1. Dazu wird die Permutation der Blätter einfach auf eine Standardvariante festgelegt.

Lemma 5.2.3 *Es sei eine beliebige zulässige (nicht notwendigerweise optimale) Lösung des ganzzahligen linearen Programms 5.1.1 gegeben. Dann existiert eine äquivalente Lösung mit:*

$$y_{v,i} = \begin{cases} 1 & \text{falls } i = v + n - 1 \\ 0 & \text{sonst} \end{cases}$$

Beweis

Der Beweis ist konstruktiv und ist nach den Erkenntnissen von Beispiel 5.2.1 beinahe offensichtlich. Da sowohl die Belegung der $y_{v,i}$ Variablen der ursprünglichen Lösung als auch die neue Belegung eine Bijektion zwischen der Knotenmenge des Graphens $V(G)$ und der Menge der Blätter des Zerlegungsbaumes $\mathcal{L}(T) = \{n, \ldots, 2 \cdot n - 1\}$

definieren, existiert eine Permuatition $\pi \in S_n$, so dass $y_{v,\pi^{-1}(i)} = 1$ in der ursprünglichen Lösung genau dann gilt, wenn $i = v + n - 1$ ist. Mit Hilfe dieser Permutation wird nun die Existenz der Lösung mit der behaupteten Eigenschaft gezeigt:

Die Baumstruktur der Lösung - d.h. die Belegung der $x_{i,j}$-Variablen - wird dabei übernommen, lediglich die $j \in \mathcal{L}(T)$ werden durch $\pi(j)$ ersetzt. Dies definiert eine zulässige Belegung der $x_{i,j}$-Variablen. Werden die Werte der $y_{v,i}$ nun zu $y_{v,\pi(i)}$ geändert, so ist dies nach Konstruktion von π konsistent, bzw. zulässig im Sinne des ganzzahligen linearen Programms. Die Belegung der $z_{v,i}$-Variablen ist nun nur noch für die Blätter entsprechend anzupassen, so dass $z_{v,i} = y_{v,i}$ für alle $v \in V(G)$ und für alle $n \leq i \leq 2 \cdot n - 1$ erfüllt ist. Die verbleibenden Werte der $z_{v,i}$ Variablen, d.h. für alle $1 \leq i \leq n - 1$, ist genau wie in der ursprünglichen Lösung zu wählen. Alle Nebenbedingungen sind dann nach Konstruktion erfüllt und der Zielfunktionswert - respektive der Wert der Variable boolw - ist gleich, damit ist eine Lösung mit der geforderten Eigenschaft gefunden. □

Im Sinne der Operation der Symmetrischen Gruppe S_n auf den Lösungen des ganzzahligen linearen Programms bedeutet das letzte Lemma, dass die Belegung der $y_{v,i}$ auf den Standardvertreter mit $y_{v,i} = 1 \Leftrightarrow i = v + n - 1$ festgelegt wird. Dies hat einige sehr positive Konsequenzen: Einerseits wird der potenzielle Lösungsraum um den Faktor $\frac{1}{n!}$ kleiner. Andererseits sind die insgesamt n^2 vielen $y_{v,i}$-Variablen damit komplett überflüssig, da implizit die Belegung aus Lemma 5.2.3 verwendet werden kann.

Das hat zur Konsequenz, dass die Ungleichungen (5.5), (5.6) und (5.7) komplett wegfallen, da sie durch die implizite Belegung der $y_{v,i}$ überflüssig werden. Die Nebenbedingung (5.8) ist anschließend nur noch zu ändern in $z_{v,i} + z_{w,i} \leq 1$, sowie zusätzlich

$$z_{v,i} = \begin{cases} 1 & \text{falls } i = v + n - 1 \\ 0 & \text{sonst} \end{cases} \tag{5.14}$$

für alle $v \in V(G)$ und für alle $n \leq i \leq 2 \cdot n - 1$ hinzuzufügen. Zusammenfassend ergibt sich so eine weitere Ersparnis von $n^2 + n$ vielen Nebenbedingungen. Doch nun soll auf das Problem aus Beispiel 5.2.2 eingegangen werden.

Lemma 5.2.4 *Es sei eine beliebige zulässige (nicht notwendigerweise optimale) Lösung des ganzzahligen linearen Programms 5.1.1 gegeben. Dann existiert eine äquivalente Lösung mit*

$$\sum_{v \in V(G)} z_{v,i} \geq \sum_{v \in V(G)} z_{v,i+1} \tag{5.15}$$

für alle $i = 1, \ldots, n - 2$.

Beweis

Gesucht wird eine Permutation π von $(1, \ldots, n)$, so dass alle $n-1$ Ungleichungen (5.15) erfüllt sind. Die Existenz dieser Permutation folgt, indem $(1, \ldots, n)$ gemäß der Regel $\mathsf{i} < \mathsf{j} :\Leftrightarrow \sum_{v \in V(G)} z_{v,\mathsf{i}} > \sum_{v \in V(G)} z_{v,\mathsf{j}}$ absteigend sortiert wird (offensichtlich gilt hier $\pi(1) = 1$). Setze nun π auf alle $\mathsf{i} = 1, \ldots, 2 \cdot n - 1$ durch $\pi(\mathsf{i}) := \mathsf{i}$ für alle $\mathsf{i} > n$ fort.

Wird nun die Belegung von $x_{\mathsf{i},\mathsf{j}}$ und $z_{v,\mathsf{i}}$ durch $x_{\pi(\mathsf{i}),\pi(\mathsf{j})}$ bzw. $z_{v,\pi(\mathsf{i})}$ ersetzt, so ergibt sich eine Lösung des ganzzahligen linearen Programms mit der gewünschten Eigenschaft. □

Hier ist anzumerken, dass die Ungleichungen (5.15) natürlich auch für $\mathsf{i} \geq n$ trivialerweise gelten, da die letzten n Knoten die Blätter des Zerlegungsbaumes sind. Weiter gilt, dass die Ungleichungen (5.14) wesentlich mehr Symmetrien vermeiden als es die Nebenbedingungen (5.15) tun. Dies liegt daran, dass es für größere Graphen noch viele Symmetrien in der Nummerierung der inneren Knoten gibt, die durch die Ungleichungen (5.15) nicht ausgeschlossen werden. Hier sind die Nebenbedingungen (5.14) wesentlich besser, sowohl von der Anzahl der Symmetrien, die vermieden werden, als auch von der Vereinfachung her, die sie ergeben. Daher ist nun noch das verbesserte ganzzahlige lineare Programm 5.2.1 angegeben. Die Korrektheit folgt sofort aus Hauptsatz 5.1.18 sowie den beiden Lemmata 5.2.3 und 5.2.4.

5.2.1 Verbessertes ganzzahliges lineares Programm zur Boolschen Weite

$$\min \mathsf{boolw}$$

$$\text{s.t. } |UN(A)| \cdot \left(\sum_{\mathsf{v} \in A} \mathsf{z_{v,i}} + \sum_{\mathsf{v} \in \overline{A}} (1 - \mathsf{z_{v,i}}) - (n-1) \right) \leq \mathsf{boolw} \qquad \forall A \subseteq V(G),\ \forall 1 \leq \mathsf{i} \leq 2n-1$$

$$\sum_{\mathsf{i}=1}^{\mathsf{j}-1} \mathsf{x_{i,j}} = 1 \qquad \forall 2 \leq \mathsf{j} \leq 2n-1$$

$$\sum_{\mathsf{j}=\mathsf{i}+1}^{2n-1} \mathsf{x_{i,j}} = 2 \qquad \forall 1 \leq \mathsf{i} \leq n-1$$

$$\sum_{\mathsf{j}=\mathsf{i}+1}^{2n-1} \mathsf{x_{i,j}} = 0 \qquad \forall n \leq \mathsf{i} \leq 2n-1$$

$$\sum_{\mathsf{v} \in V(G)} (\mathsf{z_{v,i}} - \mathsf{z_{v,i+1}}) \geq 0 \qquad \forall 1 \leq \mathsf{i} \leq n-2$$

$$\mathsf{z_{v,i}} + \mathsf{z_{w,i}} \leq 1 \qquad \forall \mathsf{v} \neq \mathsf{w} \in V(G),\ \forall n \leq \mathsf{i} \leq 2n-1$$

$$\mathsf{z_{v,v+n-1}} = 1 \qquad \forall \mathsf{v} \in V(G)$$

$$\mathsf{x_{i,j}} + \mathsf{z_{v,j}} - \mathsf{z_{v,i}} \leq 1 \qquad \forall \mathsf{v} \in V(G),\ \forall 1 \leq \mathsf{i} < \mathsf{j} \leq 2n-1$$

$$\mathsf{x_{i,j}} + \mathsf{x_{i,k}} + \mathsf{z_{v,i}} - \mathsf{z_{v,j}} - \mathsf{z_{v,k}} \leq 2 \qquad \forall \mathsf{v} \in V(G),\ \forall \mathsf{i} < \mathsf{j}, \mathsf{k},\ \mathsf{j} \neq \mathsf{k}$$

$$\mathsf{x_{i,j}} \in \{0,1\} \qquad \forall 1 \leq \mathsf{i}, \mathsf{j} \leq 2n-1$$

$$\mathsf{z_{v,i}} \in \{0,1\} \qquad \forall \mathsf{v} \in V(G),\ \forall 1 \leq \mathsf{i} \leq 2n-1$$

$$\mathsf{boolw} \in \mathbb{N}$$

Verbessertes Ganzzahliges lineares Programm für die Boolsche Weite eines Graphen G

5.3 Heuristische Lösungs- & Verbesserungsverfahren

Ein sehr großer Vorteil von ganzzahliger linearer Optimierung ist, dass verschiedene Formulierungen für diverse schwere Probleme sich aus deren eigentlichen Kontext lösen und sich ausschließlich auf eine zu optimierende Zielfunktion beschränken, so dass sich auf dieser abstrakteren Ebene viele Lösungsmöglichkeiten ergeben, die auf den ersten Blick gar nicht ersichtlich sind. So ergibt sich zwar ein mächtiges Hilfsmittel, allerdings werden so bestimmte algorithmische Gegebenheiten bisweilen nicht erfasst, die aber durchaus sehr hilfreich zur Lösung des Problems sein können.

Um diesen Problematiken etwas entgegenzuwirken, können problemspezifische Heuristiken verwendet werden, um einerseits gute zulässige Lösungen zu finden oder um andererseits bereits gegebene Lösungen heuristisch zu verbessern. Beide Varianten von heuristischen Verfahren sollen auch für das ganzzahlige lineare Programm 5.2.1 zur Bestimmung der Boolschen Weite herangezogen werden.

Vorab ist allerdings anzumerken, dass sich in Hvidevold u. a. [2011] bereits intensiver damit auseinander gesetzt wurde, in wie weit mit heuristischen Verfahren die Boolsche Weite eines Graphen zu bestimmt werden kann. Das Hauptaugenmerk liegt dort allerdings nicht darauf, einen optimalen vollständigen Zerlegungsbaum zu finden. Vielmehr wird dort versucht, heuristisch gute obere Schranken an die Boolsche Weite diverser Graphen zu bestimmen. Die Frage, ob diese auch optimal sind, wird leider nicht beantwortet. Dennoch sind die dort vorgestellten heuristischen Methoden ein guter Ansatz für Heuristiken, die beim Lösen des ganzzahligen linearen Programms zur Bestimmung der Boolschen Weite verwendet werden können.

Listing 5.2: Algorithmus `GreedySol`

```
1  GreedySol(G) {
2     /* Zu G wird eine zulässige Lösung des ILPs erstellt */
3     n = |V(G)|;
4     ILPSolution sol;
5     for i=1 to n-2 do
6        sol→x[i,i+1] = 1;
7        if (i<n-2)
8           sol→x[i,n-1+i] = 1;
9     sol→x[n-1,2n-1] = 1;
10
11    for v=1 to n do
12       sol→z[v,n-1+v] = 1;
13       for i=1 to min(v,n-1) do
14          sol→z[v,i] = 1;
15    sol→boolw = max_{2≤i≤n-1} |UN({ v ∈ V(G) : z_{v,i} = 1 })|;
16
17    return sol;
18 }
```

Begonnen werden soll mit der Heuristik `GreedySol` aus Listing 5.2, die die konkrete Struktur des Graphen ignoriert und lediglich einen vollständigen Zerlegungsbaum bzw. eine zulässige Lösung nach dem Greedy-Verfahren konstruiert, indem im Wurzelknoten des Zerlegungsbaumes ein Knoten v aus $V(G)$ entfernt wird und mit $V(G) - \{v\}$ rekursiv verfahren wird, bis alle Blätter genau einen Knoten aus G enthalten.

Zu `GreedySol` ist zu sagen, dass die so konstruierte Lösung zwar einerseits im Grunde trivial ist, aber dennoch sofort eine erste zulässige Lösung darstellt, die zur Optimierung daher hilfreich ist. Für bestimmte Graphen ist die so berechnete Lösung sogar sofort optimal, Beispiele hierfür sind Cliquen oder Pfade, wie der Beweis von Satz 3.4.4 zeigt. Darüber hinaus ist die so berechnete Lösung auch ein guter Startpunkt für heuristische Verfahren, die eine bestehende Lösung erfordern, um diese zu verbessern. Doch dazu später mehr.

Hier soll nun noch eine Heuristik vorgestellt werden, die über Graph-algorithmische Techniken einen vollständigen Zerlegungsbaum konstruiert, der anschließend in eine Lösung für das ganzzahlige lineare Programm umgewandelt wird. Der Algorithmus zur Konstruktion beruht dabei auf einer vereinfachten Variante der Algorithmen aus Hvidevold u. a. [2011], verzichtet wird hier auf nachträgliches heuristisches Optimieren der Lösung, da diese Aufgabe in erster Linie durch die Lösungsalgorithmen für das ganzzahlige lineare Programm übernommen werden sollen.

Listing 5.3: Algorithmus `DecompTreeHeuristic`

```
1  DecompTreeHeuristic() {
2     /* Zu G wird heuristisch ein vollständiger Zerlegungsbaum berechnet
3      * und daraus eine zulässige ILP-Lösung konstruiert
4      */
5
6     n = |V(G)|;
7
8     /* konstruiere Binärbaum T mit Wurzelknoten 'root' */
9     BinTree T;
10    root = T.addNode();
11    δ(root) = V(G);
12
13    while (∃ leaf t ∈ T with |δ(t)| > 1) do
14       split(t, T, δ);
15
16    /* erstelle nun eine ILP-Lösung aus (T, δ) */
17
18    /* sortiere die Knoten aus T nach absteigender Kardinalität von δ
19     * insbesondere sind die Blätter die letzten n Knoten
20     */
21    Nodes = sort([t ∈ T], (i,j) → |δ(i)| ≥ |δ(j)|);
22
23    /* erstelle neue Lösung und setze die Variablen gemäß (T, δ) */
24    ILPSolution sol;
25
26    for (source, target) ∈ A(T) do
27       i = find(source, Nodes);
28       j = find(target, Nodes);
29
30       for v=1 to n do
31          if (v ∈ δ(source))
32             sol→z[v,i] = 1;
33       if (j ≤ n-1)
34          sol→x[i,j] = 1;
35       else
36          /* ist j ≥ n so gilt δ(target) = {v} für ein v ∈ V(G) */
37          v = δ(target)[1];
38
39          sol→x[i,v+n-1] = 1;
40          sol→z[v,v+n-1] = 1;
41
42    sol→boolw = max_{t∈T} |UN(δ(t))|;
43    return sol;
44 }
```

Listing 5.4: Algorithmus `DecompTreeHeuristic::split`

```
1  DecompTreeHeuristic::split(t, T, δ) {
2     /* t ist ein Blatt des Zerlegungsbaumes T zu G sowie δ : T → Pot(V)
3      * zu t wird ein Partition (A,B) von δ(t) mit min(|A|, |B|) ≥ |δ(t)| / 3 erstellt
4      * und zwei Söhne s1, s2 von t mit δ(s1) = A und δ(s2) = B zu T hinzugefügt
5      */
6
7     P = δ(t);
8
9     /* wähle eine zufällig Teilmenge A1 ⊂ P mit ⌊P / 2⌋ Knoten */
10    while ( |A1| > |P|/2 ) do
11       i = random() mod |P|; // wähle zufälligen Index i ∈ {1, ..., |P|}
12       v = P[i+1];           // damit ist v ∈ P zufällig gewählt
13       if (v ∉ A)
14          A1 = A1 ∪ {v};
15
16    i=1;
17    while ( |P - Ai| > |P|/3 ) do
18       /* finde x in P-Ai so dass max{ UN(Ai ∪ x), UN((P\Ai)\{x}) } minimiert wird */
19       UNmax = ∞; // speichert zum minimierenden x den größeren der beiden Werte
20
21       for x ∈ P - Ai do
22          if ( max( UN(Ai ∪ {x}), UN((P\Ai)\{x}) ) < UNmax )
23             xmin = x;
24             UNmax = max( UN(Ai ∪ {x}), UN((P\Ai)\{x}) );
25
26       Ai+1 = Ai ∪ {xmin};
27       i = i+1;
28
29    /* finde i=imin so dass max { UN(Ai), UN(P - Ai) } minimiert wird */
30    UNmax = ∞; // speichert zum minimierenden i den größeren der beiden Werte
31    for j=1 to i do
32       if ( max( UN(Aj), UN(P - Aj) ) < UNmax )
33          imin = j;
34          UNmax = max( UN(Aj), UN(P - Aj) );
35
36    /* füge schließlich noch Söhne s1, s2 von t zu T hinzu */
37    s1 = T.addNode();
38    s2 = T.addNode();
39    T.addArc(t,s1);
40    T.addArc(t,s2);
41    δ(s1) = Aimin;
42    δ(s2) = P - Aimin;
43 }
```

Diese Heuristik liefert relativ gute Lösungen, für Statistiken hierzu sei ebenfalls an Hvidevold u. a. [2011] verwiesen. Es sei aber nochmals angemerkt, dass die dortigen Resultate sich auf erweiterte Verfahren beziehen, die nachträglich versuchen die Lösung heuristisch noch zu verbessern.

Listing 5.5: Algorithmus `RandomImproveHeuristic`

```
1  RandomImproveHeuristic(sol) {
2     /* Zur ILP-Lösung 'sol' wird eine neue Lösung 'new_sol' berechnet */
3
4     /* konstruiere einen Zerlegungsbaum mit 2n-1 Knoten */
5     n = |V(G)|;
6     BinTree T;
7     for i=1 to 2·n-1 do
8        i = T.addNode();
9
10    /* (i,j) ∈ A(T) ⇔ x_{i,j}=1 in sol  */
11    for i=1 to 2·n-1 do
12       for j=i+1 to 2·n-1 do
13          if (sol→x[i,j] == 1)
14             T.addArc(i,j);
15
16    /* bestimmte den Wert |δ|(i) = |δ(i)| für alle i aus sol */
17    for i=1 to n-1 do // iteriere über alle Knoten die kein Blatt sind
18       |δ|(i) = 0;
19       for j=n to 2·n-1 do // iteriere über alle Blätter
20          if (j ∈ BreadthFirstSearch(T,i))
21             |δ|(i) = δ̄(i) + 1;
22          if (t==1)
23             |δ|(j) = 1;
24
25    /* fülle die Mengen δ(i) für alle i nun heuristisch mit Knoten aus G
26     * wichtig: |δ|(i) = |δ(i)| ist jeweils bekannt
27     */
28    BETTER_SOLUTION_FOUND=false;
29    for i=1 to MAX_NUMBER_OF_ITERATIONS while (!BETTER_SOLUTION_FOUND) do
30       split(1, T, δ, |δ|);
31       if (max_{2≤i≤n-1} |UN(δ(i))| < sol→boolw)
32          BETTER_SOLUTION_FOUND=true;
33
34    /* wenn ein besserer Zerlegungsbaum gefunden wurde wandle diesen in eine ILP-Lösung um */
35    if (BETTER_SOLUTION_FOUND)
36       ILPSolution new_sol;
37       for i=1 to n-1 do
38          for j=i+1 to n-1 do
39             if ( (i,j) ∈ A(T) )
40                new_sol→x[i,j] = 1;
41          for v=1 to n do
42             if ( v ∈ δ(i) )
43                new_sol→z[v,i] = 1;
44       for j=n to 2·n-1 do
45          v=δ(j)[1];      // j ist ein Blatt, d.h. δ(j) = {v}
46          i=T.father(j); // j hat einen eindeutigen Vater i in T
47          new_sol→x[i,v+n-1]=1;
48          new_sol→z[v,v+n-1]=1;
49       new_sol→boolw = max_{2≤i≤n-1} |UN(δ(i))|;
50       return new_sol;
51    else
52       return sol;
53 }
```

Listing 5.6: Algorithmus `RandomImproveHeuristic::split`

```
1  RandomImproveHeuristic::split(t, T, δ, |δ|) {
2     /* zu gegebenem t ∈ T mit δ(t) und Söhnen s1, s2
3      * bestimme δ(s1) und δ(s2) zufällig mit Hilfe von |δ|(st) = |δ(st)|
4      */
5     (s1, s2) = T.sons(t);
6     if (|δ|(s1) > |δ|(s2))
7        swap(s1, s2);
8
9     /* wähle |δ|(s1) Knoten zufällig aus δ(t) */
10    randomVerticies = ∅;
11
12    while (|randomVerticies| < |δ|(s1)) do
13       i = random() mod |δ|(t); // wähle zufälligen Index i ∈ {1, ..., |δ|(t)}
14       v = δ(t)[i+1];          // damit ist v ∈ δ(t) zufällig gewählt
15       if (v ∉ randomVerticies)
16          randomVerticies = randomVerticies ∪ {v};
17
18    /* füge nun die zufällig gewählten Knoten entweder zu δ(s1) oder δ(s2) hinzu */
19    for v ∈ randomVerticies do
20       if (random() mod 2 == 0)
21          δ(s1) = δ(s1) ∪ {v};
22       else
23          δ(s2) = δ(s2) ∪ {v};
24
25    /* füge nun alle verbleibenden Knoten in δ(s1) oder δ(s1) ein */
26    for v ∈ δ(t)-randomVerticies do
27       if (δ(s1) == |δ|(s1))      // δ(s1) hat keine Kapazität mehr
28          δ(s2) = δ(s2) ∪ {v};
29       else if (δ(s2) == |δ|(s2)) // δ(s2) hat keine Kapazität mehr
30          δ(s1) = δ(s1) ∪ {v};
31       else                       // δ(s1) und δ(s2) haben noch Kapazitäten
32          if ( UN(δ(s1) ∪ {v}) < UN(δ(s2) ∪ {v}) )
33             δ(s1) = δ(s1) ∪ {v};
34          else if ( UN(δ(s2) ∪ {v}) < UN(δ(s1) ∪ {v}) )
35             δ(s2) = δ(s2) ∪ {v};
36          else if ( |δ(s1)| < |δ(s2)| )
37             δ(s1) = δ(s1) ∪ {v};
38          else
39             δ(s2) = δ(s2) ∪ {v};
40
41    /* ist s1 bzw. s2 kein Blatt arbeite rekursiv auf t=s1 bzw. t=s2 */
42    if ( |δ(s1)| > 1 )
43       split(s1, T, δ, |δ|);
44
45    if ( |δ(s2)| > 1 )
46       split(s2, T, δ, |δ|);
47 }
```

Einen anderen Ansatz verfolgen die Heuristiken aus den Listings 5.5 bzw. 5.6. Hier wird die Strategie verfolgt, eine zulässige Lösung des ganzzahligen linearen Programms zufallsgesteuert zu verändern und so möglicherweise eine bessere Lösung zu erhalten. Dazu wird die Baumstruktur - d.h. die Belegung der $x_{i,j}$-Variablen - weitestgehend übernommen und eine andere Bijektion zwischen den Blättern des Zerlegungsbaumes und den Knoten des Graphen bestimmt. Nach den Ergebnissen des letzten Abschnitts heißt das konkret, dass eine Permutation der Blätterindizes bestimmt wird und anschließend nur noch die Variablen entsprechend angepasst werden müssen, so dass sich eine zulässige Lösung ergibt.

Algorithmus `RandomImproveHeuristic` arbeitet damit nach dem Prinzip, dass im Wesentlichen die Größe der Mengen $|\delta(i)|$ zu den Knoten $i \in T$ übernommen wird und mit Hilfe von Zufallsentscheidungen und lokaler Minimierung der UN-Werte die Mengen $\delta(i)$ neu bestimmt werden. Begonnen wird bei dem Wurzelknoten, dessen δ-Label nach Definition ganz $V(G)$ ist.

Unter Verwendung von `RandomImproveHeuristic::split` wird damit das Label der beiden Söhne des Wurzelknotens - und völlig analog über rekursive Aufrufe das Label aller Knoten - heuristisch bestimmt. Dieses Verfahren wird so lange iteriert, bis eine bessere Lösung gefunden oder eine bestimmte obere Grenze von Iterationen erreicht wird. Gesteuert wird dies über den Wert der Konstanten `MAX_NUMBER_OF_ITERATIONS`.

Mit Hilfe der drei vorgestellten Heuristiken ist gewährleistet, dass relativ schnell gute Lösungen gefunden werden, was sich unmittelbar in entsprechenden primalen Schranken, während des Lösens des ganzzahligen linearen Programms, widerspiegelt. Bevor eine Auswertung und Analyse der Laufzeiten und Ergebnisse erfolgt, wird nun noch auf die genaueren Details der Implementierung eingegangen.

5.4 Details der Implementierung

Da bisher genaue Details zur Implementierung der vorgestellten Programme und Funktionen sowie des Zusammenspiels zwischen den einzelnen Teilen ausgeblendet wurden, folgt nun noch ein Abschnitt, der sich mit eben diesen Punkten beschäftigt. Mit dem MIP-Inferface von Kutschka u. a. [2012] ist ein sehr nützliches Werkzeug gegeben, um eigenen C++-Programmcode in Lösungssoftware wie SCIP oder CPLEX einzubinden. Entsprechend wurden alle Implementierungen unter Verwendung des MIP-Inferface in C++ vorgenommen.

Natürlich muss zu Beginn zunächst der Ausgangsgraph G eingelesen und verarbeitet werden. Hierzu wurde die sehr umfangreiche und kostenlose Open-Source LEMON Graph Library verwendet. Diese wird nun kurz vorgestellt, weitere Informationen sind in Jüttner u. a. [2010], sowie online unter `http://lemon.cs.elte.hu/`, zu finden.

5.4.1 Die LEMON Graph Library und das LEMON Graph Format

Für den hier benötigten Zweck stellt die LEMON Graph Library mit dem LEMON Graph Format zunächst eine sehr gute Möglichkeit dar, Graphen zu speichern und zu verarbeiten. In den angefertigten Implementierungen wird zunächst der Graph G eingelesen, der daher im LEMON Graph Format vorliegen muss. Grundsätzlich bietet die LEMON Graph Library aber auch Kompatibilität zu anderen Graph-Formaten, wie etwa dem DIMACS Graph Format. Dazu ist allerdings dann der entsprechende Funktionsaufruf zum Einlesen des jeweiligen Formates anzupassen. Da in den angefertigten Implementierungen aber das LEMON Graph Format verwendet wird, soll dies hier kurz vorgestellt werden.

Ein Graph im LEMON Graph Format besteht aus zwei Teilen, die jeweils den Knoten und Kanten entsprechen. Diese werden mit einem `@nodes` bzw. `@edges` gekennzeichnet und anschließend erfolgt eine Aufzählung der Knoten bzw. Kanten. Für gerichtete Graphen, die hier allerdings nicht betrachtet werden[2], ist `@edges` durch `@arcs` zu ersetzen. Jede Zeile des jeweiligen Abschnitts entspricht dabei genau einem Knoten bzw. einer Kante, Abbildungen auf Knoten und Kanten (z.B. Distanzen, Kapazitäten oder Kosten) können in beliebiger Anzahl in jeder Zeile hinzugefügt werden, indem die Datei um eine weitere Spalte, deren Überschrift den Namen der Abbildung trägt, erweitert wird.

Die Werte der Abbildung für Knoten v bzw. Kanten e können dann in der entsprechenden Spalte der jeweiligen Zeile zu v bzw. e eingetragen werden. Diese Abbildungen sind allerdings bis auf Ausnahme optional: Jeder Knoten und jede Kante muss ein Label haben. Hier ist es weder notwendig, dass das Label bei 1 anfängt und bei $n = |V(G)|$ bzw. $m = |E(G)|$ endet, noch ist es erforderlich, dass die Labels aufsteigend oder absteigend sind - hier sind beliebige Permutationen möglich. Allerdings ist es für eine bessere Übersicht (insbesondere in größeren Graphen) natürlich vorteilhafter, wenn die Knoten von 1 bis n und die Kanten von 1 bis m untereinander stehen, weshalb dies auch in allen hier betrachteten Beispielen stets der Fall ist.

[2] Gerichtete Graphen spielen nur eine Rolle in den Implementierungen der Heuristiken, um dort Zerlegungsbäume zu repräsentieren.

Für den hier benötigten Zweck reichen diese Punkte auch schon aus um alle benötigten Eigenschaften zu beschreiben.

Listing 5.7: Beispiel für das LEMON Graph Format

```
1  # Beispiel für einen Graph im lgf-Format mit 6 Knoten und 8 Kanten
2
3  @nodes
4  label
5  1
6  2
7  3
8  4
9  5
10 6
11
12 @edges
13           label
14 1    2    1
15 1    3    2
16 1    5    3
17 2    3    4
18 2    6    5
19 3    4    6
20 4    5    7
21 4    6    8
```

In Listing 5.7 werden die angesprochenen Punkte veranschaulicht. In Zeile 3 bis 10 werden sechs Knoten mit Label 1 bis 6 definiert, in Zeile 12 bis 21 werden insgesamt acht Kanten mit Label 1 bis 8 angegeben. Die ersten beiden Spalten in der Definition einer Kante beziehen sich dabei auf die Labels der Knoten, die durch diese Kante verbunden werden. Insbesondere müssen also die Labels der Knoten, die durch die jeweilige Kante verbunden werden, immer existieren (andernfalls scheitert das Einlesen des Graphen). Eine Datei mit Namen `example.lgf` und dem Inhalt aus Listing 5.7 kann dann, mit den entsprechenden Funktionen der LEMON Graph Library, eingelesen werden.

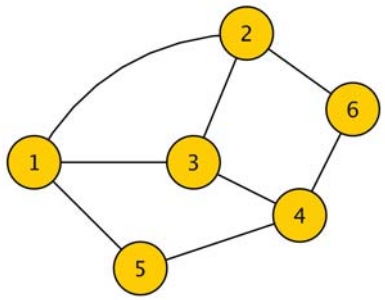

Abbildung 5.4: Das Beispiel zum LEMON Graph Format aus Listing 5.7.

Mit Hilfe des LEMON Graph Format ist damit sowohl eine Möglichkeit gegeben, die Graphen zu speichern und zu bearbeiten, als auch um diese unter C++ verwenden zu können.

Zur weiteren Verarbeitung der Graphen wird dann der Graphtyp `ListGraph` gewählt, der einen Standardtyp für ungerichtete Graphen darstellt. Für weitere Details hierzu, sowie andere von LEMON unterstützte Graphtypen, sei an dieser Stelle direkt auf die LEMON-Dokumentation verwiesen. Entscheidend ist nur, dass somit alle benötigten Graphoperationen zur Verfügung gestellt werden.

5.4.2 Konstruktion des ganzzahligen linearen Programms

Da mit Hilfe der LEMON Graph Library und den Ausführungen des letzten Abschnitts eine gute Möglichkeit für den Umgang von C++ mit Graphen gegeben ist, soll nun darauf eingegangen werden, wie alle Bestandteile des ganzzahligen linearen Programms aus Abschnitt 5.1 zusammengefügt werden. Wie oben bereits angesprochen, ist das entscheidende Werkzeug hierzu das MIP-Inferface von Kutschka u. a. [2012], das direktes Anbinden aller Bestandteile an Lösungssoftware wie SCIP oder CPLEX ermöglicht.

Das Anlegen der Variablen $x_{i,j}$, $z_{v,i}$ und boolw ist dabei, basierend auf den dafür zur Verfügung gestellten Funktionen, selbsterklärend. Im Folgenden wird daher auf das Einbinden der Nebenbedingungen und der Heuristiken aus Abschnitt 5.3 eingegangen.

Die Nebenbedingungen sind dazu in eine Normalform

$$\texttt{lhs} \leq \sum_{\texttt{i}} \texttt{coeffs[i]} \cdot \texttt{vars[i]} \leq \texttt{rhs}$$

zu überführen, wobei die Spezialfälle $\texttt{lhs} = -\infty$, $\texttt{rhs} = \infty$ und $\texttt{lhs} = \texttt{rhs}$ zugelassen sind. Für die Werte ∞ und $-\infty$ dienen dabei Konstanten, die von der jeweiligen Lösungssoftware abhängen. Um die Nebenbedingungen einzubinden sind also - abhängig von der jeweiligen Situation - die Konstanten `lhs` und `rhs` anzupassen und Vektoren `coeffs` sowie `vars` anzulegen, deren Inhalt entsprechend zu wählen ist. Dies ist für alle Nebenbedingungen des ganzzahligen linearen Programms problemlos machbar, einzig den Nebenbedingungen der Form

$$|UN(A)| \cdot \left(\sum_{v \in A} z_{v,i} + \sum_{v \in \overline{A}} (1 - z_{v,i}) - (|V(G)| - 1) \right) \leq \text{boolw}$$

kommt hierbei noch eine besondere Rolle zu, da hier einerseits der Ausdruck noch zur Normalform umgeformt werden muss und andererseits diese zwangsläufig vom Wert $|UN(A)|$ abhängt.

Zunächst bedarf es hier eines Formates um Mengen von Knoten effizient zu verwalten. Daher werden die Knoten von G intern als $v \in \{1, \ldots, n\}$ behandelt, realisiert wird das in C++ durch zwei Abbildungen

```
map<int, ListGraph::Node> Number_to_Vertex        und
map<ListGraph::Node, int> Vertex_to_Number.
```

Somit können die Knoten des Graphen zu jeder Zeit als $v \in \{1, \ldots, n\}$ behandelt werden, insbesondere beim Benennen der Variablen und Nebenbedingungen. Für graphentheoretische Operationen auf G, wie z.B. dem Berechnen von Nachbarschaften, sind dann, mit Hilfe von `Number_to_Vertex`, die jeweilige Knoten in G zu bestimmen und die Ergebnisse anschließend, unter Verwendung von `Vertex_to_Number`, zurück zu übertragen auf $\{1, \ldots, n\}$. Der Zeitaufwand dieser Abbildungen ist pro Aufruf nur logarithmisch in der Anzahl aller Elemente, es ergibt sich also nur ein Aufwand von $\log(n)$ pro Aufruf einer dieser Abbildungen.

Da die Knoten von $G = (V, E)$ nun als $v \in \{1, \ldots, n\}$ angenommen werden können, bietet es sich an, alle potenziellen $A \subseteq V$ als $A \subseteq \{1, \ldots, n\}$ aufzufassen. Dann kann A als Vektor vom Typ `vector<int>` in C++ verarbeitet werden, was einerseits relativ effizient ist, andererseits aber keine mengentheoretischen Funktionen wie Schnitt oder Vereinigung zur Verfügung stellt.

Weiterhin ist in C++ die Gleichheit von Vektoren nur über Gleichheit aller Elemente und gleicher Länge implementiert. Entsprechend muss jeder Vektor, der eine Menge $A \subseteq V$ repräsentieren soll, zwingend sortiert sein, um Gleichheitsabfragen korrekt entscheiden zu können. Zusätzliche benötigte Funktionen können darauf aufbauend nach Bedarf modelliert werden, ob ein Element im jeweiligen Vektor enthalten ist oder nicht. Dies realisiert die C++-Funktion `find`, die einen Pointer auf das gesuchte Element liefert (bzw. auf das Ende des Vektors, falls das Element im Vektor nicht vorkommt). Somit ist eine Gleichheitsabfrage und eine Element-Enthalten-Abfrage gegeben, womit alle weiteren benötigten Funktionen modelliert werden können.

Nun kann, für gegebenes A, der Wert $|UN(A)|$ mit Hilfe von Algorithmus 5.1 bestimmt werden. Zu beachten ist hier aber, dass einerseits $|UN(A)| = |UN(\overline{A})|$, nach Satz 4.2.9, gilt und andererseits der Wert $|UN(A)|$ nicht nur beim Verwalten der Nebenbedingungen wichtig ist, sondern auch in den in Abschnitt 5.3 vorgestellten Heuristiken benötigt wird. Es ist daher sinnvoll, den Wert $|UN(A)|$ - und damit automatisch auch $|UN(\overline{A})|$ - nur einmal zu berechnen und anschließend beide Werte abzuspeichern. Sollte dann zu einem späteren Zeitpunkt ein bereits bekannter Wert angefragt werden, so kann dieser direkt zurückgegeben werden.

Hier wurde dies mit einer Abbildung `map<vector<int>, int> stored_UN` in C++ umgesetzt. Diese bedarf exponentiellem Speicher, da grundsätzlich alle 2^n Knotenmengen abgespeichert werden könnten. In jeder Suchanfrage ergibt sich aber nur logarithmischer Zeitaufwand in der Größe der Abbildung, entsprechend liefert dies insgesamt nur einen Zeitaufwand $\mathcal{O}(n)$.

Festzuhalten bleibt damit, dass in den obigen Nebenbedingungen der Wert $|UN(A)|$ stets als bekannt vorauszusetzen ist. Somit müssen die Nebenbedingungen nur noch auf die Normalform für das MIP-Interface gebracht werden:

$$\begin{aligned}
& |UN(A)| \cdot \left(\sum_{v \in A} z_{v,i} + \sum_{v \in \overline{A}} (1 - z_{v,i}) - (|V| - 1) \right) && \leq \mathsf{boolw} \\
\iff & |UN(A)| \cdot \left(\sum_{v \in A} z_{v,i} - \sum_{v \in \overline{A}} z_{v,i} + \sum_{v \in \overline{A}} 1 - |V| + 1 \right) && \leq \mathsf{boolw} \\
\iff & |UN(A)| \cdot \left(\sum_{v \in A} z_{v,i} - \sum_{v \in \overline{A}} z_{v,i} - |A| + 1 \right) && \leq \mathsf{boolw} \\
\iff & \sum_{v \in A} |UN(A)| \cdot z_{v,i} + \sum_{v \in \overline{A}} -|UN(A)| \cdot z_{v,i} - \mathsf{boolw} && \leq |UN(A)| \cdot (|A| - 1)
\end{aligned}$$

So können auch diese Nebenbedingungen für alle $A \subseteq V$ und für alle $i = 1, \ldots, 2 \cdot n - 1$ eingebunden werden. Benötigt wird nur dazu noch ein einfacher Algorithmus, der alle $A \subseteq V$ in Form eines `vector< vector<int> >` bestimmt, was rekursiv aber sehr einfach gelöst werden kann.

Anzumerken ist noch, dass die Einschränkung $i = 1, \ldots, n - 1$ gemacht werden kann: Insofern der Graph mindestens eine Kante hat, liefert jedes Blatt maximal die Bedingung $\mathsf{boolw} \geq 2$ und falls $E = \emptyset$ ist, so folgt schon $\mathsf{boolw} = 1$ und das Problem wird mit jedem vollständigen Zerlegungsbaum sofort optimal gelöst. Ist also $E \neq \emptyset$, genügt eine Nebenbedingung $\mathsf{boolw} \geq 2$ um alle $i = n, \ldots, 2 \cdot n - 1$ abzudecken. Ebenso kann $i = 1$ vernachlässigt werden, da die Wurzel nur $\mathsf{boolw} \geq 1$ liefert. Somit verbleiben diese Nebenbedinungen für alle $A \subseteq V$ und für alle $i = 2, \ldots, n - 1$, es gibt also insgesamt $2^n \cdot (n-2)$, und damit exponentiell viele, solche Ungleichungen.

5.4.2.1 Lazy-Constraints

Zur Verarbeitung eines ganzzahligen linearen Programms, das u.a. exponentiell viele Nebenbedingungen eines bestimmten Typs hat, bietet es sich an, diese im Lösungsprozess zunächst zu ignorieren und eine optimale Lösung für das ganzzahlige lineare

Programm zu bestimmen, das sich ergibt, wenn diese Ungleichungen entfernt werden. Ist dann eine optimale Lösung für das sich ergebende ganzzahlige lineare Programm gefunden, so muss diese nur zulässig für die ignorierten Nebenbedingungen sein.

Wird dann eine verletzte Nebenbedingung gefunden, so kann diese dem ganzzahligen linearen Programm hinzugefügt werden und dieses erneut gelöst werden. Falls aber keine verletzten Nebenbedingungen gefunden werden, so ist eine Optimallösung für das ursprüngliche Problem gefunden. Natürlich muss hier beachtet werden, wie eine solche verletzte Ungleichung gefunden wird, insofern sie existiert. Auch dabei muss vermieden werden, alle diese Nebenbedingungen anzugeben, denn sonst ergeben sich wieder sofort exponentiell viele und in der Summe ist nichts gewonnen. Entsprechende Methoden hängen natürlich stark vom betrachteten Modell ab und sollen hier, für das konstruierte ganzzahlige lineare Programm 5.2.1 für die Boolsche Weite eines Graphen, vorgestellt werden.

Für den Umgang mit Lazy-Constraints bietet das MIP-Interface eine Klasse an, in der zwei Funktionen zu implementieren sind: Die erste Funktion `is_feasible` entscheidet, ob eine übergebene Lösung zulässig ist, d.h. es wird geprüft, ob es verletzte Nebenbedingungen gibt. Falls dem so ist wird die zweite Funktion `separate` aufgerufen, in der verletzte Ungleichungen dem ganzzahligen linearen Programm hinzugefügt werden können.

Listing 5.8: `is_feasible`-Funktion zum Umgang mit Lazy-Constraints

```
1  is_feasible(sol) {
2     /* prüft ob die ILP-Lösung 'sol' Lazy-Constraints verletzt oder nicht */
3
4     for i=2 to n-1 do
5        A = { v ∈ V(G) : sol→z[v,i] = 1 };
6
7        if ( |UN(A)| > sol→boolw )
8           /* verletzte Nebenbedingung gefunden */
9           return false;
10
11    /* wurde keine verletzte Nebenbedingung gefunden ist 'sol' zulässig */
12    return true;
13 }
```

Listing 5.8 zeigt die `is_feasible`-Funktion. Die Schleife iteriert hier über alle Knoten des Baumes, die nicht die Wurzel oder ein Blatt sind, da für diese höchstens boolw ≥ 2 folgt, was bereits erfüllt ist. Nun soll hier noch die Funktion `separate` vorgestellt werden, die verletzte Nebenbedingungen zum ganzzahligen linearen Programm hinzufügt. Ebenso wie `is_feasible` überprüft auch `separate` die Wurzel und die Blätter des Zerlegungsbaumes aus genannten Gründen nicht.

Listing 5.9: `separate`-Funktion zum Umgang mit Lazy-Constraints

```
1  separate(sol) {
2     /* findet verletzte Nebenbedingungen zur ILP-Lösung 'sol' und fügt diese dem ILP hinzu */
3
4     number_of_cuts = 0;
5     for i=2 to n-1 do
6        A = { v ∈ V(G) : sol→z[v,i] = 1 };
7        Ā = { v ∈ V(G) : sol→z[v,i] = 0 };
8
9        if ( |UN(A)| > sol→boolw )
10          addCut(∑_{v∈A} |UN(A)|·z[v,i] + ∑_{v∈Ā} -|UN(A)|·z[v,i] - boolw ≤ |UN(A)|·(|A| - 1));
11          number_of_cuts = number_of_cuts + 1;
12
13    return number_of_cuts;
14 }
```

Somit kann zwar die explizite Formulierung von exponentiell, genauer $2^n \cdot (n-2)$, vielen Nebenbedingungen vermieden werden, nur besteht jetzt keine direkte Verbindung zwischen dem Wert der Variable boolw und der restlichen Variablenbelegung mehr. Das ist insofern schlecht für die Lösbarkeit des ganzzahligen linearen Programms, da nun die Lösungssoftware eine zulässige Belegung für die $x_{i,j}$ und $z_{v,i}$ findet und zusätzlich den Wert der Variablen boolw auf den kleinst möglichen Wert setzt, da dies gleichzeitig die zu minimierende Zielfunktion ist. Damit verwendet die Software leider relativ viel Zeit darauf, viele verschiedene zulässige Lösungen zu finden, in denen jeweils die Zielfunktionsvariable boolw minimal gewählt wird, und die nachfolgend alle separiert werden.

Daher dauert es leider selbst für kleine Graphen relativ lange, bis Ungleichungen der Form boolw $\geq c$, für geeignete Konstanten $c > 2$, gefolgert werden können. Auf weitere Details zu diesen Punkten wird in der Auswertung der Ergebnisse in Abschnitt 5.5 eingegangen.

Vor diesem Hintergrund wurde daher in den angefertigten Implementierungen die Möglichkeit gegeben, über den Wert einer boolschen Variable `use_LazyCons` zu steuern, ob die entsprechenden Nebenbedingungen implizit, mit Hilfe von `separate` und `is_feasible`, verwaltet werden oder explizit dem ganzzahligen linearen Programm hinzugefügt werden sollen. Für die implizite Handhabung ist `use_LazyCons = true` zu wählen, mit `use_LazyCons = false` werden die Ungleichungen explizit hinzugefügt.

5.4.2.2 Einbinden der Heuristiken

Ein wichtiger Bestandteil des Lösungsprozesses des vorgestellten ganzzahligen linearen Programms 5.2.1 sind die Heuristiken aus Abschnitt 5.3. In welcher Form diese, mit Hilfe des MIP-Interface, an die Lösungssoftware angebunden werden, soll hier noch kurz erklärt werden.

Das MIP-Interface stellt zum Einbinden von Heuristiken eine eigene Klasse zur Verfügung, in der im Wesentlichen nur eine Funktion `run` zu implementieren ist. Diese bekommt die beste bekannte ganzzahlige Lösung des ganzzahligen linearen Programms übergeben, falls eine solche bekannt ist, und berechnet eine neue, möglicherweise bessere Lösung, die über einen Pointer zurück übermittelt wird. Ob die beste bekannte Lösung nun in der Heuristik verwendet wird oder nicht, ist dabei alleine von der Funktion `run` abhängig. Entsprechend können hierüber gleichermaßen alle in Abschnitt 5.3 vorgestellten Heuristiken angebunden werden.

Ursprünglich wurden in Abschnitt 5.3 die drei Heuristiken `GreedySol` (Algorithmus 5.2), `DecompTreeHeuristic` (Algorithmus 5.3) und `RandomImproveHeuristic` (Algorithmus 5.5) vorgestellt. Dabei sind die ersten beiden komplett unabhängig von einer besten bekannten Lösung des ganzzahligen linearen Programms, während die letzte auf diese unbedingt angewiesen ist.

Zum Anbinden der Heuristiken wurde daher eine Klasse `DecompTreeHeuristic`, die genau der Implementierung von Algorithmus 5.3 in C++ entspricht, und eine Klasse `ImproveHeuristic`, in der sowohl Algorithmus 5.2 als auch Algorithmus 5.5 eingebunden wurden, erstellt. Beim Aufruf der Funktion `ImproveHeuristic::run` wird unterschieden, ob eine beste bekannte ganzzahlige Lösung übergeben wurde oder nicht. Ist dies nicht der Fall, so kann es sich nur um den ersten Aufruf der Funktion überhaupt handeln, denn anschließend ist durch `GreedySol` mindestens eine ganzzahlige Lösung bekannt, folglich auch mindestens eine beste.

Wird daher keine beste bekannte ganzzahlige Lösung übergeben, so wird hier eine solche mit `GreedySol` erstellt. Andernfalls wird `RandomImproveHeuristic` ausgeführt, um möglicherweise eine bessere Lösung zu bestimmen. Alle weiteren Details ergeben sich im Grunde direkt aus den Algorithmen 5.2, 5.3 und 5.5, es sind nur noch die Hilfsfunktionen `split` (siehe Algorithmus 5.4 und 5.6) zu implementieren und an C++ anzupassen.

Als Datenstruktur für vollständige Zerlegungsbäume diente hier, wie bereits weiter oben kurz angedeutet, die Klasse `ListDigraph`, die ebenfalls in der LEMON Graph Library zu finden ist, zusammen mit einer Abbildung δ. Die Zerlegungsbäume wurden

als gerichtete Graphen implementiert, um die Vater/Sohn-Relation auf den Knoten repräsentieren zu können, d.h. die Wurzel hat als einziger Knoten keine eingehenden Kanten während, die Blätter keine ausgehenden besitzen. Alle anderen Knoten haben genau eine eingehende Kante, die von ihrem Vater kommt, und zwei ausgehende, die jeweils zu ihren Söhnen führen.

In vollständiger Analogie zu den beiden Abbildungen (s.o.) `Number_to_Vertex` und `Vertex_to_Number`, mit denen die Knoten von G als $\{1,\ldots,n\}$ angenommen werden konnten, wurden hier zwei Abbildungen `Number_to_Node` und `Node_to_Number` definiert, mit denen die Knoten des vollständigen Zerlegungsbaumes als $\{1,\ldots,2\cdot n-1\}$ angenommen werden können, vgl. Lemma 5.1.1. Entsprechend ist die Abbildung δ in der Funktion `ImproveHeuristic::run` auch als `map< int, vector<int> >` implementiert, wobei sortierte Vektoren des Typs `vector<int>` wieder Mengen repräsentieren. Einzig in der Funktion `DecompTreeHeuristic::run` wurde die Abbildung δ direkt auf den Knoten, d.h. als `map< ListDigraph::Node, vector<int> >`, verwendet, da hier heuristisch die Mengen konstruiert werden und anschließend erst die Knoten des Zerlegungsbaumes mit Zahlen gelabelt werden. Für weitere Details sei hier direkt auf den Quellcode verwiesen.

Abschließend gibt es noch eine boolsche Variable `use_Heuristics`, mit der gesteuert werden kann, ob die Heuristiken dem ganzzahligen linearen Programm hinzugefügt werden sollen oder nicht. Dies ist ebenfalls analog zum Parameter `use_LazyCons`, der im vorherigen Abschnitt vorgestellt wurde, zu sehen. Für die meisten Fälle ist es allerdings vorteilhaft, die Heuristiken zu verwenden.

5.5 Ergebnisse und Analyse

Das vorgestellte ganzzahlige lineare Programm soll nun abschließend noch auf Laufzeit untersucht und der Frage nachgegangen werden, in wie weit dies eine praktikable Methode ist, um die Boolsche Weite eines gegebenen Graphen zu bestimmen. Als Lösungssoftware wurden dabei sowohl das Open-Source-Projekt SCIP 2.1.1 als auch die kommerzielle Software CPLEX 12.4.0.0 aus dem Hause IBM verwendet, gerechnet wurde auf jeweils einem 2.93 GHz Quad-Core CPU mit 12 GB Arbeitsspeicher unter Linux.

Mit Blick auf das ganzzahlige lineare Programm 5.2.1 für die Boolsche Weite in der verbesserten Version ergeben sich für einen Graphen mit $|V(G)| = n$ insgesamt $\mathcal{C}(n) = (n-2)2^n + \frac{8}{3}n^4 - 5n^3 + \frac{10}{3}n^2 + 5n - 5$ Nebenbedingungen. Selbst wenn Lazy-Constraints (vgl. Teilabschnitt 5.4.2.1) verwendet werden, verbleiben daher also immer noch $\mathcal{C}_L(n) = \frac{8}{3}n^4 - 5n^3 + \frac{10}{3}n^2 + 5n - 5$ Stück. Tabelle 5.1 zeigt die Größenordnung der beiden Folgen.

Tabelle 5.1: Größenordnung der beiden Folgen $\mathcal{C}(n)$ und $\mathcal{C}_L(n)$

$n =$	**10**	**15**	**20**	**25**	**30**	**40**	**50**
$\mathcal{C}(n) \approx$	$30 \cdot 10^3$	$550 \cdot 10^3$	$19 \cdot 10^6$	$772 \cdot 10^6$	$30 \cdot 10^9$	$41 \cdot 10^{12}$	$54 \cdot 10^{15}$
$\mathcal{C}_L(n) \approx$	$22 \cdot 10^3$	$120 \cdot 10^3$	$390 \cdot 10^3$	$970 \cdot 10^3$	$2 \cdot 10^6$	$6.5 \cdot 10^6$	$16 \cdot 10^6$

Entsprechend ist es nicht allzu verwunderlich, dass der Lösungsprozess leider selbst auf diesem starken Rechner bereits bei kleineren Graphen ziemlich viel Zeit in Anspruch nimmt. Daher wurde für die Laufzeitanalyse keine bereits verfügbare Graphdatenbank verwendet. Statt dessen wurden einige kleinere Graphen getestet, die nun zunächst vorgestellt werden sollen.

Einerseits wurden die quadratischen Gittergraphen G_3, G_4 und G_5 sowie deren Verallgemeinerungen, d.h. die rechteckigen Gittergraphen $G_{3\times4}$, $G_{3\times5}$ und $G_{4\times5}$, verwendet. Um an die Aussagen von Kapitel 4 anzuknüpfen wurden diese erweitert um die H-Gitter HG_3, HG_4 und HG_5 sowie die I-Gitter IG_3, IG_4 und IG_5. Schließlich wurden noch zwei weitere Graphen verwendet, nämlich der berühmte Petersen-Graph PS und der Pyramid3-Graph PY_3. Beide sind in Abbildung 5.5 dargestellt.

Das ganzzahlige lineare Programm wurde mit der jeweiligen Lösungssoftware sowohl mit als auch ohne Lazy-Constraints getestet. Die Heuristiken aus Abschnitt 5.3 wurden dabei stets verwendet, allerdings nur in den ersten 100 Knoten des Branch-&-Bound-Baumes sowie anschließend nur noch alle 100 Knoten. Bei den beiden Durchläufen, in denen keine Lazy-Constraints verwendet wurden, musste aber leider auf die Analyse der Graphen $G_{4\times5}$, G_5, HG_5 und IG_5 verzichtet werden, da hier die Erstellung des ganzzahligen linearen Programms wegen zu viel Speicherbedarf nicht möglich war, vgl. Tabelle 5.1.

In allen Durchläufen wurde die Laufzeit dabei auf eine Stunde begrenzt. Insofern das ganzzahlige lineare Programm bis dahin also noch keine optimale Lösung gefunden hatte, wurde die letzte Ausgabe vor Ende der Laufzeit als Referenz zu einer Stunde Laufzeit angegeben. Die Ergebnisse von SCIP sind in den Tabellen 5.2 und 5.3 zu sehen, die von CPLEX in den Tabellen 5.4 und 5.5.

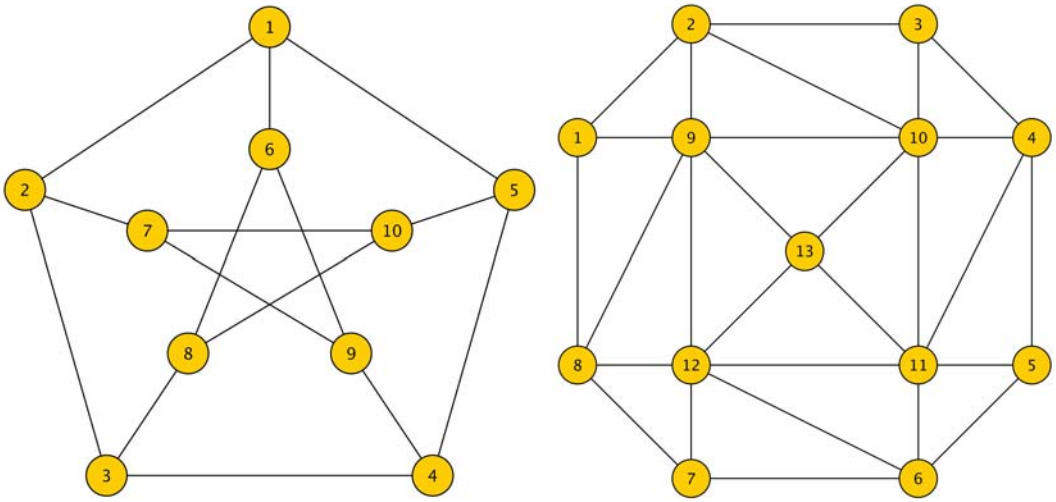

Abbildung 5.5: Der berühmte Petersen- und der Pyramid3-Graph

Bemerkenswert ist hier insbesondere, dass SCIP fast in allen Fällen bessere Ergebnisse als CPLEX liefert. Alle Durchläufe zeigen unabhängig davon aber im Wesentlichen die gleichen Probleme auf. Es werden zwar - nicht zuletzt dank der relativ guten Heuristiken, insbesondere bei kleineren Graphen - ziemlich schnell gute Lösungen gefunden, allerdings sind mangelnde gute duale Schranken das zentrale Problem.

Dies ist insofern nicht verwunderlich, da die Zielfunktion nur durch eine Variable gegeben ist und alle anderen Variablen lediglich einen zulässigen Wert haben müssen. Es wird daher versucht, die Zielfunktionsvariable boolw auf dem kleinst möglichen Wert zu halten und eine zulässige Belegung für die $x_{i,j}$- und $z_{v,i}$-Variablen zu finden. Wird boolw allerdings zu klein gewählt - was anfangs fast immer der Fall ist - dauert es entsprechend lange bis festgestellt wird, dass keine zulässige Lösung für die anderen Variablen existiert. Verschärft wird dies noch, wenn Lazy-Constraints benutzt werden, da dann zunächst gar keine Verbindung zwischen der Zielfunktion und der Belegung der $x_{i,j}$ und $z_{v,i}$ besteht.

Entscheidend verbessern ließe sich diese Problematik, und damit auch die Laufzeit des ganzzahligen linearen Programms, wenn schnell zu bestimmende, untere Schranken an die Boolsche Weite bekannt wären, die eine Ungleichung $\text{boolw} \geq c$, für ein hinreichend gewähltes $c > 2$, bedeuten würden. Diese Fragestellung führt direkt zu Kapitel 3, wodurch zumindest gezeigt ist, dass die meisten unteren Schranken an die Baumweite ausscheiden. Mit einer derartigen Verbesserung würden wahrscheinlich wesentlich bessere Laufzeiten erzielt und wesentlich mehr Graphen gelöst werden können.

Festzuhalten bleibt hier allerdings, dass zumindest in der Theorie, so die Boolsche Weite eines beliebigen Graphen bestimmt werden kann, wie die entsprechenden Aussagen dieses Kapitels zeigen.

Tabelle 5.2: Ergebnisse von SCIP mit Heuristiken und ohne Lazy-Constraints.

	time	node	it	mem	frac	vars	cons	cols	rows	cuts	du-bd	pr-bd	gap
G_3	1.8s	1	469	30M	55	162	8011	162	8026	24	2.0	5.0	1.5
	24.1s	166	6252	31M	50	162	8060	162	8009	24	4.0	4.0	0.0
G_4	2345s	1	1368	3290M	205	568	974k	568	974k	20	2.0	14.0	6.0
	3600s	1	1368	3290M	205	568	974k	568	974k	20	2.0	14.0	6.0
$G_{3\times4}$	111s	1	843	183M	93	306	57k	306	57k	29	2.0	6.0	2.0
	3600s	11k	479k	276M	84	306	58k	306	57k	29	2.0	5.0	1.5
$G_{3\times5}$	3461s	1	1804	1547M	191	495	469k	495	469k	33	2.0	10.0	4.0
	3600s	1	1804	1547M	191	495	469k	495	469k	33	2.0	10.0	4.0
HG_3	2.6s	1	356	31M	59	162	8011	162	8026	24	2.0	3.0	0.5
	43.2s	1182	21223	32M	–	162	8841	0	0	24	3.0	3.0	0.0
HG_4	2302s	1	1194	3286M	196	568	974k	568	974k	12	2.0	4.0	1.0
	3600s	1	1194	3286M	196	568	974k	568	974k	12	2.0	4.0	1.0
IG_3	2.7s	1	358	32M	55	162	8011	162	8030	28	2.0	3.0	0.5
	22.9s	103	4439	32M	64	162	7999	162	8012	28	3.0	3.0	0.0
IG_4	2413s	1	1711	3291M	206	568	974k	568	974k	21	2.0	4.0	1.0
	3600s	1	1711	3291M	206	568	974k	568	974k	21	2.0	4.0	1.0
PS	3.0s	1	648	54M	63	205	15k	205	15k	29	2.0	8.0	3.0
	3600s	117k	1903k	316M	–	205	25k	0	0	29	4.0	8.0	1.0
PY_3	176s	1	1099	371M	111	364	113k	364	113k	24	2.0	10.0	4.0
	3600s	3400	175k	447M	89	364	113k	364	113k	24	2.0	8.0	3.0

Tabelle 5.3: Ergebnisse von SCIP mit Heuristiken und Lazy-Constraints.

	time	node	it	mem	frac	vars	cons	cols	rows	cuts	du-bd	pr-bd	gap
G_3	1.9s	1	318	22M	65	315	4569	315	4569	24	2.0	5.0	1.5
	464s	34k	651k	23M	–	315	5387	315	4570	31k	4.0	4.0	0.0
G_4	29.7s	1	1201	224M	126	1064	57k	1064	57k	45	2.0	9.0	3.5
	3600s	14k	965k	291M	–	1064	59k	0	0	19k	2.0	7.0	2.5
G_5	1110s	1	1676	1394M	275	2675	381k	2675	381k	25	2.0	30.0	14.0
	3600s	8	21k	1399M	554	2675	381k	2675	381k	49	2.0	18.0	8.0
$G_{3\times4}$	9.3s	1	539	68M	85	582	16k	582	16k	35	2.0	6.0	2.0
	3600s	63k	2601k	184M	–	582	19k	582	16k	45k	4.0	5.0	0.25
$G_{3\times5}$	24.2s	1	852	173M	119	930	43k	930	43k	31	2.0	10.0	4.0
	3600s	16k	1050k	209M	111	930	44k	930	43k	14k	2.0	5.0	1.5
$G_{4\times5}$	442s	1	1480	568M	216	1690	149k	1690	149k	16	2.0	18.0	8.0
	3600s	1000	202k	596M	147	1690	150k	1690	149k	662	2.0	9.0	3.5
HG_3	1.9s	1	318	22M	65	315	4569	315	4569	24	2.0	3.0	0.5
	13.6s	283	5944	22M	–	315	4747	315	4573	234	3.0	3.0	0.0
HG_4	29.7s	1	1201	224M	126	1064	57k	1064	57k	45	2.0	4.0	1.0
	3600s	14k	867k	261M	104	1064	59k	1064	57k	12k	2.0	4.0	1.0
HG_5	1108s	1	1676	1394M	275	2675	381k	2675	381k	25	2.0	5.0	1.5
	3600s	1	1676	1394M	275	2675	381k	2675	381k	25	2.0	5.0	1.5
IG_3	1.9s	1	352	22M	64	315	4569	315	4576	31	2.0	3.0	0.5
	10.3s	168	3202	22M	–	315	4662	315	4561	141	3.0	3.0	0.0
IG_4	30.0s	1	1201	224M	126	1064	57k	1064	57k	45	2.0	4.0	1.0
	3600s	13k	830k	248M	113	1064	59k	1064	57k	13k	2.0	4.0	1.0
IG_5	1107s	1	1676	1394M	275	2675	381k	2675	381k	25	2.0	5.0	1.5
	3600s	1	1676	1394M	275	2675	381k	2675	381k	25	2.0	5.0	1.5
PS	1.6s	1	435	32M	59	395	7371	395	7395	50	2.0	8.0	3.0
	3600s	163k	3938k	164M	–	395	14k	395	7379	123k	4.0	8.0	1.0
PY_3	9.1s	1	641	94M	95	689	23k	689	23k	39	2.0	10.0	4.0
	3600s	34k	1956k	183M	–	689	24k	0	0	29k	2.0	8.0	3.0

Tabelle 5.4: Ergebnisse von CPLEX mit Heuristiken und ohne Lazy-Constraints.

	Time	Node	Obj.	B. Int.	B. Bd.	ItCnt	Gap	Variable B	NodeID	Parent	Depth
G_3	1.3s	0	2.0	6.0	2.0	129	0.66	–	0	–	0
	593.6s	232k	4.0	4.0	4.0	1862k	0.0	–	–	–	–
G_4	2225.2	0	2.0	14.0	2.0	1206	0.85	–	0	–	0
	3600s	2	2.0	12.0	2.0	9857	0.83	z_10_7 D	2	1	2
$G_{3\times4}$	24.1s	0	2.0	9.0	2.0	507	0.77	–	0	–	0
	3600s	48k	4.0	5.0	2.0	815k	0.6	z_3_8 D	47690	47689	33
$G_{3\times5}$	406.5s	0	2.0	18.0	2.0	623	0.88	–	0	–	0
	3600s	1176	2.0	6.0	2.0	46k	0.66	z_5_5 D	1176	1175	35
HG_3	1.5s	0	2.0	3.0	2.0	138	0.33	–	0	–	0
	64.9s	21k	3.0	3.0	3.0	212k	0.0	–	–	–	–
HG_4	1532.7s	0	2.0	4.0	2.0	885	0.5	–	0	–	0
	3600s	2	2.0	4.0	2.0	11k	0.5	z_15_2 U	2	0	1
IG_3	1.5s	0	2.0	3.0	2.0	138	0.33	–	0	–	0
	610s	259k	3.0	3.0	3.0	1886k	0.0	–	–	–	–
IG_4	2485.3s	0	2.0	4.0	2.0	1189	0.5	–	0	–	0
	3600s	1	2.0	4.0	2.0	9365	0.5	z_12_11 D	1	0	1
PS	3.6s	0	2.0	8.0	2.0	260	0.75	–	0	–	0
	3600s	414k	2.0	8.0	2.0	6509k	0.75	z_3_8 U	413816	413812	35
PY_3	85.5s	0	2.0	10.0	2.0	671	0.8	–	0	–	0
	3600s	14k	5.0	8.0	2.0	389k	0.75	z_2_7 U	13990	13989	34

Tabelle 5.5: Ergebnisse von CPLEX mit Heuristiken und Lazy-Constraints.

	Time	Node	Obj.	B. Int.	B. Bd.	ItCnt	Gap	Variable B	NodeID	Parent	Depth
G_3	0.7s	0	2.0	6.0	2.0	157	0.66	–	0	–	0
	849.3s	380k	4.0	4.0	4.0	4782k	0.0	–	–	–	–
G_4	61.3s	0	2.0	12.0	2.0	1323	0.83	–	0	–	0
	3600s	31k	2.0	8.0	2.0	1698k	0.75	z_1_15 U	31498	31469	56
G_5	2992.2s	0	2.0	16.0	2.0	8260	0.875	–	0	–	0
	3600s	1	2.0	16.0	2.0	8311	0.875	z_19_15 D	1	0	1
$G_{3\times4}$	9.2s	0	2.0	6.0	2.0	659	0.66	–	0	–	0
	3600s	223k	4.0	5.0	2.0	4418k	0.6	x_5_1 U	223371	223370	129
$G_{3\times5}$	58.8s	0	2.0	7.0	2.0	1764	0.71	–	0	–	0
	3600s	76k	2.0	6.0	2.0	2644k	0.66	z_15_12 D	75978	75977	52
$G_{4\times5}$	651.3s	0	2.0	18.0	2.0	3330	0.88	–	0	–	0
	3600s	4993	2.0	9.0	2.0	382k	0.77	z_5_8 D	4993	4992	150
HG_3	0.6s	0	2.0	3.0	2.0	146	0.33	–	0	–	0
	1007.6s	380k	3.0	3.0	3.0	6099k	0.0	–	–	–	–
HG_4	78.6s	0	2.0	4.0	2.0	1711	0.5	–	0	–	0
	3600s	38k	2.0	4.0	2.0	1757k	0.5	z_2_14 D	37500	37499	37
HG_5	1022s	0	2.0	5.0	2.0	3773	0.6	–	0	–	0
	3600s	1	2.0	5.0	2.0	3962	0.6	z_14_15 D	1	0	1
IG_3	0.6s	0	2.0	3.0	2.0	146	0.33	–	0	–	0
	940.7s	371k	3.0	3.0	3.0	5485k	0.0	–	–	–	–
IG_4	48.9s	0	2.0	4.0	2.0	1321	0.5	–	0	–	0
	3600s	38k	2.6	4.0	2.0	1704k	0.5	z_15_11 U	38482	38481	85
IG_5	3600s	0	2.0	5.0	2.0	11k	0.6	–	–	–	–
	3600s	0	2.0	5.0	2.0	11k	0.6	–	–	–	–
PS	1.2s	0	2.0	8.0	2.0	193	0.75	–	0	–	0
	3600s	640k	4.0	8.0	2.0	15497k	0.75	x_5_11 D	640011	640010	46
PY_3	11.9s	0	2.0	9.0	2.0	660	0.77	–	0	–	0
	3600s	156k	2.0	8.0	2.0	3437k	0.75	z_11_5 U	156349	149472	61

6 Fazit, kritische Bewertung und offene Probleme

Werden die Ergebnisse insgesamt rückblickend bewertet, so ist zu Kapitel 3 zu sagen, dass sich leider gültige untere Schranken an die Baumweite im Wesentlichen nicht auf Boolsche Weite übertragen lassen. Einzig die beiden Eigenschaften, dass die Baumweite eines nicht zusammenhängenden Graphen der maximalen Baumweite seiner Zusammenhangskomponenten entspricht, sowie, dass die Baumweite eines induzierten Teilgraphen eine untere Schranke an die Baumweite des ursprünglichen Graphen ist, ließen sich auf Boolsche Weite übertragen. Anzumerken ist hier allerdings, dass die Baumweite eines beliebigen Teilgraphen, und allgemeiner der eines beliebigen Minors, eine untere Schranke an die Baumweite eines Graphen ist, während dies für die Boolsche Weite nicht zutrifft.

Alle weiteren vorgestellten Schranken lassen sich leider, in der aktuellen Formulierung, in keinster Weise auf Boolsche Weite übertragen. Da durch die vollständigen Graphen K_n jeweils eine Familie von Graphen gegeben ist, deren jeweilige Boolsche Weite konstant ist, während ihre Baumweite in $\Theta(n)$ liegt, ist auch gezeigt, dass diese Schranken selbst logarithmisch keine unteren Schranken an die Boolsche Weite sind. Dies wäre insofern interessant, da nach Definition der Boolschen Weite der Ausdruck $\max_{t \in T} \log_2 |UN(\delta(t))|$ über alle vollständigen Zerlegungsbäume zu minimieren ist und somit auch untere Schranken an den Wert $\min_{(T,\delta) \in \mathcal{T}_f(G)} \max_{t \in T} |UN(\delta(t))| = 2^{boolw(G)}$ von Interesse wären, insbesondere für das in Kapitel 5 vorgestellte ganzzahlige lineare Programm.

Allerdings liefert hier die Cliquenweite, nach Satz 3.4.20, eine untere Schranke an den Wert $\min_{(T,\delta) \in \mathcal{T}_f(G)} \max_{t \in T} |UN(\delta(t))|$ und somit ist letztendlich jede untere Schranke an die Cliquenweite auch eine gültige untere Schranke an $2^{boolw(G)+1}$. Da die Cliquenweite, ebenfalls als unmittelbare Folgerung aus Satz 3.4.20, einer Familie von Graphen beschränkt ist, wenn es die Boolsche Weite ist, ist ebenfalls gezeigt, dass alle unteren Schranken an die Baumweite, die keine Gültigkeit für Boolsche Weite haben, auch keine Gültigkeit für Cliquenweite besitzen. In diesem Sinne wären hier auch weitere untere Schranken an die Cliquenweite interessant, um daraus untere Schranken an die Boolsche Weite abzuleiten.

Ein Beispiel dafür wäre die Rangweite $rw(G)$: Nach Belmonte und Vatshelle [2011] und Bui-Xuan u. a. [2009] gilt nämlich $\log_2(rw(G))-1 \leq \log_2(cw(G))-1 \leq boolw(G)$, d.h. eine untere Schranke an die Rangweite liefert eine logarithmische untere Schranke an die Boolsche Weite. Ein Algorithmus, der eine untere Schranke an die Rangweite berechnet, wird u.a. in Beyß [2012] vorgestellt. Dies wäre also ein möglicher Ansatz, um weitere untere Schranken für die Boolsche Weite herzuleiten, die evtl. auch zur Verbesserung weiterer Ergebnisse dienen könnten.

Etwas positiver fällt das Fazit zu Kapitel 4 aus. Hier ist es allem voran gelungen, die Boolsche Weite der $(k \times k)$-Gittergraphen G_k zu bestimmen, da nach Hauptsatz 4.3.22 $boolw(G_k) \approx 0.81137 \cdot k - 0.46968$ mit einem Fehler $\overset{k\to\infty}{\longrightarrow} 0$ gilt. Dieses offene Problem aus Bui-Xuan u. a. [2009] und Hvidevold u. a. [2011] konnte damit weitestgehend gelöst werden. So ist außerdem gezeigt, dass die Heuristiken für die Boolsche Weite aus Hvidevold u. a. [2011] auf Gittergraphen sehr gute Ergebnisse liefern. Die gezeigte Verallgemeinerung $boolw(G_k) \leq boolw(G_{k\times\ell}) \leq boolw(G_{k+1})$ auf $(k\times\ell)$-Gittergraphen mit $\ell \geq k$ aus Hauptsatz 4.3.26 sei hier ebenfalls angemerkt.

Besonders interessant sind die Ergebnisse zu den H-Gittern HG_k und I-Gittern IG_k, vgl. Definition 3.4.21 und 4.5.1. Nach Hauptsatz 4.4.2 gilt hier für die Boolsche Weite $\log_2(k)-1 \leq boolw(HG_k) \leq \log_2(k)$, womit sie bei dieser Graphklasse fast endgültig bestimmt ist. Eine wünschenswerte Verbesserung wäre es hier, die vermutlich exakte obere Schranke $boolw(HG_k) \leq \log_2(k)$ auch als untere Schranke zu erhalten.

Etwas schlechter sehen die Ergebnisse für die I-Gitter IG_k aus. Hier ergibt sich mit Hauptsatz 4.5.5 ein Bereich $\log_2(k)-1 \leq boolw(IG_k) \leq 2\cdot\log_2(k)-1$, wobei die obere Schranke nach Lemma 4.5.4 sogar noch minimal besser, und auch hier vermutlich optimal ist, so dass sich hier ebenfalls die Frage nach einer besseren unteren Schranke aufdrängt.

Insofern eine minimal bessere untere Schranke für die I-Gitter gefunden wird, so sind nach den Einsichten von Abschnitt 4.6 auch Cliquenseparatoren nicht sicher für Boolsche Weite, was für Baumweite allerdings zutrifft. Dies führt wieder zur Frage, ob es gute untere Schranken an die Boolsche Weite gibt, die dieses Problem beheben würden.

Weiterhin ist es interessant, hier eine Antwort auf die Frage zu finden, ob überhaupt sichere Separatoren für Boolsche Weite existieren. Mit solchen Separatoren könnte die Suche nach untere Schranken für die Boolsche Weite auf kleinere Graphen reduziert werden, wo hoffentlich leichter eine Antwort zu finden ist.

Letztendlich verbleibt hier aber in beiden Fällen $boolw(HG_k) \in \Theta(\log_2(k))$ und $boolw(IG_k) \in \Theta(\log_2(k))$. Dies ist insofern bemerkenswert, da einerseits sowohl $tw(HG_k) \in \Omega(k)$ und $tw(IG_k) \in \Omega(k)$ (da beide Graphen k-Cliquen enthalten) als auch $cw(HG_k) \in \Theta(k)$ sowie $cw(IG_k) \in \Theta(k)$ gilt (vgl. Satz 3.4.23 bzw. 4.5.2).

Mit Blick auf die eingangs angesprochenen Lösungsmöglichkeiten für das maximale gewichtete stabile Mengenproblem und das minimale gewichtete Dominanzmengenproblem ergeben sich hier also zwei interessante Beispiele, in denen die Lösungsmöglichkeiten über die Boolsche Weite polynomial in k sind, während die jeweiligen Pendants über Baumweite exponentiell in k bleiben.

Es wäre interessant, hier die Boolsche Weite weiterer Graphen zu untersuchen, um so weitere Beispiele für diese Situation zu finden und somit die Relevanz der Boolschen Weite unterstreichen zu können. Mit Hilfe der dominanten Mengen bzw. Folgerung 4.2.5 ist hier ein sehr nützliches Werkzeug gegeben, um die Boolsche Weite einer Knotenmenge bzw. eines Zerlegungsbaumes - und damit auch die eines Graphen selbst - bestimmen zu können, ohne auf die konkrete Berechnung von UN-Werten zurückgreifen zu müssen.

Schließlich wurde in Kapitel 5 noch eine Möglichkeit vorgestellt, um für einen gegebenen Graphen G, mit Hilfe der ganzzahligen linearen Programmierung, die Boolsche Weite $boolw(G)$ bestimmen zu können. Allerdings funktioniert dies nur theoretisch für beliebige Graphen. Bereits für Graphen mit $10 \leq n \leq 20$ Knoten hat sich selbst die verbesserte Version 5.2.1 des ganzzahligen linearen Programms als nicht praktikabel erwiesen, für Graphen ab 20 Knoten bedarf oftmals selbst die Erstellung zu viel Speicher.

Für alle diese Problematiken gibt es in erster Linie zwei Ursachen: Schwächen in der Formulierung und mangelnde Schranken an die Zielfunktion. Konkret ist zunächst die Anzahl der Nebenbedingungen, vor allem durch die Modellierung der Zielfunktion, mit $\mathcal{C}(n) \approx (n-2) \cdot 2^n + 3 \cdot n^4$ zu groß, wobei $n = |V(G)|$ ist.

Hier ist es durch die Verwendung von Lazy-Constraints zwar möglich, die $(n-2) \cdot 2^n$ vielen Nebenbedingungen nicht explizit formulieren zu müssen, wodurch aber die Beziehung zwischen zulässiger Variablenbelegung und der Zielfunktion fast in Gänze verloren geht und erst im Lösungsprozess hergestellt werden muss. Verschärft wird diese Problematik noch durch zu viele Symmetrien in den potenziellen Lösungen, die dazu führen, dass oft mehrfach äquivalente Lösungen gefunden und auf Zulässigkeit bzw. Optimalität hin überprüft werden müssen.

Das andere Hauptproblem, das die langen Laufzeiten verursacht, sind mangelnde duale Schranken an die Zielfunktion. Insbesondere mit den zur Verfügung stehenden Heuristiken für die Boolsche Weite eines Graphen können relativ schnell gute zulässige Lösungen bzw. primale Schranken gefunden werden, allerdings mangelt es an Möglichkeiten, um die duale Schranke zu verbessern. Dies führt sofort in die offenen Probleme aus Kapitel 3, nach dem die Frage nach guten unteren Schranken an die Boolsche Weite nach wie vor offen ist. Insofern einige gute untere Schranken an die Boolsche Weite bekannt wären, könnten diese auch im vorgestellten ganzzahligen linearen Programm verwendet werden, was sich hoffentlich positiv auf die Laufzeit auswirkt, da dann optimale Lösungen erwartungsgemäß schneller erkannt werden könnten.

Bisher wären solche Methoden, für zusammenhängende Graphen, basierend auf den Ergebnissen des 3. Kapitels, nur theoretisch über die Erkennung von induzierten Teilgraphen möglich, deren Boolsche Weite bekannt ist. Dies führt allerdings unweigerlich auf das Graphisomorphieproblem und ist, in der Ermangelung von vielen Graphen mit bekannter Boolscher Weite, damit praktisch als schwierig einzustufen. Die Analyse von nicht zusammenhängenden Graphen kann bereits auf die Analyse der Komponenten eingeschränkt werden, wie insbesondere gezeigt wurde.

Eine andere Verbesserung wären starke Nebenbedingungen, die mehr Symmetrien ausschließen, als es die bisher verwendeten Ungleichungen aus Abschnitt 5.2 tun. Alternativ wäre auch eine Neuformulierung der Zielfunktion oder des gesamten ganzzahligen linearen Programms denkbar, um eine Zielfunktion zu erhalten, die hoffentlich besser geeignet ist, um eine optimale Lösung zu finden. All dies sind Ansätze und Motivationen für weitere Untersuchungen.

Literaturverzeichnis

Adler u. a. 2010
Adler, I. ; Bui-Xuan, B. M. ; Rabinovich, Y. ; Renault, G. ; Telle, J. A. ; Vatshelle, M.: On the Boolean-width of a graph: structure and applications. In: *Proceedings of the 36th international conference on Graph-theoretic concepts in computer science*. Berlin, Heidelberg : Springer-Verlag, 2010 (WG'10), S. 159–170 1

Arnborg und Proskurowski 1986
Arnborg, S. ; Proskurowski, A.: Characterization and recognition of partial 3-trees. In: *SIAM J. Algebraic Discrete Methods* 7 (1986), Nr. 2, S. 305–314 1, 3.1

Arnborg und Proskurowski 1989
Arnborg, S. ; Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. In: *Discrete Applied Mathematics* 23 (1989), Nr. 1, S. 11–24 1

Belmonte und Vatshelle 2011
Belmonte, R. ; Vatshelle, M.: Graph Classes with Structured Neighborhoods and Algorithmic Applications. 2011 1, 6

Beyß 2012
Beyss, M.: *Fast Algorithms for Rank-Width*, RWTH Aachen University, Diplomarbeit, 2012 6

Bodlaender 1988
Bodlaender, H. L.: Dynamic Programming on Graphs with Bounded Treewidth. In: *ICALP*, 1988, S. 105–118 1

Bodlaender 1998
Bodlaender, H. L.: A partial k-arboretum of graphs with bounded treewidth. In: *Theoretical Computer Science* 209 (1998), Nr. 1-2, S. 1–45 3.1

Bodlaender und Koster 2003
Bodlaender, H. L. ; Koster, A. M. C. A.: Safe Separators for Treewidth /

Department of Information and Computing Sciences, Utrecht University. 2003 (UU-CS-2003-027). – Forschungsbericht 4.6

Bodlaender und Koster 2007

Bodlaender, H. L. ; Koster, A. M. C. A.: On the Maximum Cardinality Search Lower Bound for Treewidth. In: *Discrete Applied Mathematics* 155 (2007), S. 1348–1372 3.4.4

Bodlaender und Koster 2010

Bodlaender, H. L. ; Koster, A. M. C. A.: Treewidth computations I. Upper bounds. In: *Information and Computation* 208 (2010), Nr. 3, S. 259–275 3.4

Bodlaender und Koster 2011

Bodlaender, H. L. ; Koster, A. M. C. A.: Treewidth Computations II. Lower Bounds. In: *Information and Computation* 209 (2011), Nr. 7, S. 1103–1119. – http://dx.doi.org/10.1016/j.ic.2011.04.003 3.1, 3.4, 3.4.1, 3.4.1, 3.4.12, 3.4.13, 3.4.18, 3.4.4

Bouchitté u. a. 2004

Bouchitté, V. ; Kratsch, D. ; Müller, H. ; Todinca, I.: On treewidth approximations. In: *Discrete Applied Mathematics* 136 (2004), Nr. 2-3, S. 183–196 3.4

Bui-Xuan u. a.

Bui-Xuan, B. M. ; Telle, J. A. ; Vatshelle, M.: *Fast algorithms for vertex subset and vertex partitioning problems on graphs of low boolean-width.* – Department of Informatics, University of Bergen, Norway 5

Bui-Xuan u. a. 2009

Bui-Xuan, B. M. ; Telle, J. A. ; Vatshelle, M.: Boolean-Width of Graphs. In: *Parameterized and Exact Computation*. Springer-Verlag, 2009 1, 3.4, 3.4.2, 3.4.15, 3.4.20, 4.1, 6

Corneil und Rotics 2001

Corneil, D. G. ; Rotics, U.: On the Relationship between Clique-Width and Treewidth. In: *Proceedings of the 27th International Workshop on Graph-Theoretic Concepts in Computer Science*, Springer-Verlag, 2001 (WG '01), S. 78–90 1

Courcelle u. a. 1993

Courcelle, B. ; Engelfriet, J. ; Rozenberg, G.: Handle-rewriting Hypergraph Grammars. In: *Journal of Computer and System Sciences* 46 (1993), Nr. 2, S. 218–270 3.2

Courcelle und Olariu 1997

Courcelle, B. ; Olariu, S.: Upper Bounds to the Clique-Width of Graphs. In: *Discrete Applied Mathematics* 101 (1997), S. 77–114 1, 3.4

Diestel 2000

Diestel, R.: *Graphentheorie*. Springer-Verlag Heidelberg, 2000 2

Golumbic und Rotics 2000

Golumbic, M. C. ; Rotics, U.: On the Clique-Width of Some Perfect Graph Classes. In: *Int. J. Found. Comput. Sci.* 11 (2000), Nr. 3, S. 423–443 1, 3.2, 4.3.3, 4.5, 4.5, 4.5.2

Graham u. a. 1994

Graham, R. L. ; Knuth, D. E. ; Patashnik, O.: *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., 1994 4.3.3

Hedtstück 2000

Hedtstück, U.: *Einführung in die Theoretische Informatik: Formale Sprachen und Automatentheorie*. Oldenbourg, 2000 4.3.3

Hvidevold u. a. 2011

Hvidevold, E. M. ; Sharmin, S. ; Telle, J. A. ; Vatshelle, M.: Finding Good Decompositions for Dynamic Programming on Dense Graphs. 2011 1, 4.1, 5, 5.1, 5.3, 5.3, 5.3, 6

Johansson 1998

Johansson, Ö.: Clique-Decomposition, NLC-Decomposition, and Modular Decomposition - Relationships and Results for Random Graphs. In: *Congressus Numerantium* 132 (1998), S. 39–60 1

Jüttner u. a. 2010

Jüttner, A. ; Dezső, B. ; Kovács, P.: LEMON - Library for Efficient Modeling and Optimization in Networks / Dept. of Operations Research, Eötvös Loránd University, Budapest. 2010. – Forschungsbericht 5.4

Kim 1982

Kim, K. H.: *Monographs and Textbooks in Pure and Applied*. Bd. 70: *Boolean Matrix Theory and Applications*. Marcel Dekker, 1982 4.2.9

Kloks und Bodlaender 1992

Kloks, T. ; Bodlaender, H. L.: Only few graphs have bounded treewidth /

Department of Information and Computing Sciences, Utrecht University. 1992 (RUU-CS-92-35). – Forschungsbericht 1

Kutschka u. a. 2012

Kutschka, M. ; Raack, C. ; Tieves, M.: *MIP-Interface for SCIP, CPLEX and GUROBI*. 2012 5.4, 5.4.2

Köbler u. a. 1993

Köbler, J. ; Schöning, U. ; Torán, J.: *The Graph Isomorphism Problem: Its Structural Complexity*. Basel, Switzerland, Switzerland : Birkhäuser Verlag, 1993 2.4

Lozin 2008

Lozin, V. V.: From Tree-Width to Clique-Width: Excluding a Unit Interval Graph. In: *Proceedings of the 19th International Symposium on Algorithms and Computation*. Berlin, Heidelberg : Springer-Verlag, 2008 (ISAAC '08), S. 871–882 1

Lucena 2002

Lucena, B.: *Dynamic Programming, Tree-Width, and Computation on Graphical Models*, Brown University, Providence, RI, USA, Diss., 2002 3.4.4, 3.4.30

Lucena 2003

Lucena, B.: A New Lower Bound for Tree-Width Using Maximum Cardinality Search. In: *SIAM J. Discret. Math.* 16 (2003), S. 345–353 3.4, 3.4.4, 3.4.30

Nemhauser und Wolsey 1988

Nemhauser, G. L. ; Wolsey, L. A.: *Integer and combinatorial optimization*. New York, NY, USA : Wiley-Interscience, 1988 5

Nörlund 1924

Nörlund, N. E.: *Vorlesungen über Differenzenrechnung*. Springer-Verlag Berlin, 1924 4.3.3

Olesen und Madsen 1999

Olesen, K. G. ; Madsen, A. L.: Maximal Prime Subgraph Decomposition of Bayesian Networks / Aalborg University. Aalborg, Denmark, 1999. – Forschungsbericht 4.6.2

Ramachandramurthi 1995

Ramachandramurthi, S.: A lower bound for treewidth and its consequences. In: *Graph-Theoretic Concepts in Computer Science* Bd. 903. Springer Berlin / Heidelberg, 1995, S. 14–25 3.4, 3.4.1, 3.4.8

Ramachandramurthi 1997

Ramachandramurthi, S.: The Structure and Number of Obstructions to Treewidth. In: *SIAM J. Discret. Math.* 10 (1997), S. 146–157 3.4.1

Schrijver 1986

Schrijver, A.: *Theory of Linear and Integer Programming.* Wiley-Interscience, 1986 5

Stanley 2004

Stanley, R. P.: *Combinatorics and Commutative Algebra.* Birkhäuser Boston, 2004 (Bd. 41) 4.3.3

Sunil Chandran und Subramanian 2003

Sunil Chandran, L. ; Subramanian, C. R.: A spectral lower bound for the treewidth of a graph and its consequences. In: *Information Processing Letters* 87 (2003), S. 195–200 3.4, 3.4.3, 3.4.3, 3.4.27

Sunil Chandran und Subramanian 2005

Sunil Chandran, L. ; Subramanian, C. R.: Girth and treewidth. In: *J. Comb. Theory Ser. B* 93 (2005), S. 23–32 3.4, 3.4.1, 3.4.1, 3.4.10

Takata 2010

Takata, K.: Space-optimal, backtracking algorithms to list the minimal vertex separators of a graph. In: *Discrete Applied Mathematics* 158 (2010), Nr. 15, S. 1660–1667 4.6

Tarjan und Yannakakis 1984

Tarjan, R. E. ; Yannakakis, M.: Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. In: *SIAM J. Comput.* 13 (1984), S. 566–579 3.4.4

van Rooij u. a. 2009

van Rooij, J. M. M. ; Bodlaender, H. L. ; Rossmanith, P.: Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In: *ESA* Bd. 5757, Springer-Verlag, 2009 (Lecture Notes in Computer Science), S. 566–577 1

Volkmann 2006

Volkmann, L.: *Graphen an allen Ecken und Kanten.* 2006 2

Wolsey 1998

Wolsey, L. A.: *Integer Programming.* New York, NY, USA : Wiley-Interscience, 1998 5

Eidesstattliche Erklärung

Ich, Björn Böken, Matrikelnummer 280047, versichere hiermit, dass ich meine Masterarbeit mit dem Thema

Boolsche Weite: Analyse, Schranken & Lösbarkeit

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Mir ist bekannt, dass ich meine Masterarbeit zusammen mit dieser Erklärung fristgemäß nach Vergabe des Themas in dreifacher Ausfertigung und gebunden im Prüfungsamt der RWTH Aachen einzureichen oder spätestens mit dem Poststempel des Tages, an dem die Frist abläuft, zu senden habe.

Aachen, den 25. Juni 2012

Björn Böken

Stichwortverzeichnis