# Anwendungen monadischer Logik zweiter Stufe auf Probleme beschränkter Baumweite und deren Platzkomplexität

von Christoph Stockhusen

Diese Diplomarbeit wurde von Till Tantau betreut und am 8. März 2011
am Institut für Theoretische Informatik der Universität zu Lübeck vorgelegt.

## EIDESSTATTLICHE ERKLÄRUNG

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Lübeck, 8. März 2011 _____

## ZUSAMMENFASSUNG

Die parametrisierte Komplexitätstheorie befasst sich mit der komplexitätstheoretischen Untersuchung von Problemen unter besonderer Beachtung ihrer inhärenten Parameter. Historisch bedingt liegt der Fokus dabei auf der Untersuchung der Zeitkomplexitäten von Problemen. In dieser Arbeit wird ein Framework eingeführt, welches die parametrisierte Komplexitätstheorie um Komplexitätsklassen und einen passenden Reduktionsbegriff erweitert, die erste systematische Untersuchungen bezüglich der Platzkomplexitäten von Problemen ermöglichen. Weiter wird eine Auswahl von klassischen Problemen unter Verwendung dieses Frameworks analysiert. Dies beinhaltet verschiedene Varianten, vor allem Zählvarianten, von Zahlenproblem, dem Erfüllbarkeitsproblem für aussagenlogische Formeln sowie Graphen- und Mengenproblemen, die mittels der Logspace-Versionen der Sätze von Bodlaender und Courcelle bezüglich ihrer Platzkomplexität klassifiziert werden. Abschließend gibt diese Arbeit einen Ausblick auf weiterführende Entwicklungen der parametrisierten Platzkomplexität.

# INHALTSVERZEICHNIS

# KAPITEL 1

# EINLEITUNG

Die Komplexität vieler Probleme ändert sich stark, wenn die Klasse der betrachteten Probleminstanzen so eingeschränkt wird, dass Annahmen gemacht werden können, die im Allgemeinen nicht gültig sind. So existieren Graphenprobleme, die auf Bäumen oder planaren Graphen in polynomieller Zeit berechenbar sind, während eine Berechnung des Problems auf beliebigen Graphen bisher nur in exponentieller Zeit möglich ist und nach bisherigen Erkenntnissen und Einschätzungen von Experten bezüglich der P-NP-Frage auch bleiben wird (23).

Ein oft betrachtetes Konzept zur Einschränkung von Probleminstanzen eines Graphenproblems ist die *beschränkte Baumweite*, die zusammen mit dem Konzept der *Baumzerlegung* von Robertsen und Seymour (39) eingeführt wurde. Die Baumweite eines Graphen beschreibt, wie ähnlich dieser einem Baum ist; die Klasse der Graphen mit beschränkter Baumweite beinhaltet diejenigen Graphen, die eine große Ähnlichkeit zu Bäumen haben. Die Baumzerlegung eines Graphen ist ein Baum, der durch Zerlegung des betrachteten Graphen entsteht und die baumähnliche Struktur eines Graphen darstellt. Aus der bestmöglichen Baumzerlegung eines Graphen ergibt sich dessen Baumweite.

Für Graphen mit beschränkter Baumweite lassen sich viele Probleme mittels dynamischer Programmierung unter Verwendung einer zu der Baumweite korrespondierenden Baumzerlegung effizient berechnen, wie beispielsweise das NP-vollständige Färbbarkeitsproblem mit drei Farben, das sich so in linearer Zeit entscheiden lässt (18). Die Berechnung der Baumzerlegung, die hierfür benötigt wird, lässt sich nach dem *Satz von Bodlaender* (6) für Graphen mit beschränkter Baumweite ebenfalls in linearer Zeit berechnen, während die Berechnung der Baumweite und damit der Baumzerlegung für Graphen, deren Baumweite nicht beschränkt ist, ein NP-vollständiges Problem darstellt (2).

Werden Probleme betrachtet, deren Instanzen auf Graphen mit beschränkter Baumweite begrenzt sind, so lassen sich Aussagen über deren Komplexität treffen, indem Algorithmen formuliert werden, die unter Verwendung der Baumzerlegung und der problemabhängigen Eigenschaften eine Lösung berechnen. Die Analyse des Ressourcenverbrauchs solcher Algorithmen ermöglicht Aussagen über die Komplexität des betrachteten Problems. Die Formulierung und Analyse dieser Algorithmen ist jedoch oftmals sehr komplex. Durch den *Satz von Courcelle* (14) ist es möglich, diese Vorgänge zu vereinfachen, denn dieser besagt unter der Verwendung des Satzes von Bodlaender, dass Aussagen in monadischer Logik zweiter

Stufe über Graphen mit beschränkter Baumweite in linearer Zeit entscheidbar sind.

Elberfeld, Jakoby und Tantau zeigten, dass die Sätze von Bodlaender und Courcelle auch gelten, wenn jeweils die lineare Zeitschranke durch eine logarithmische Platzschranke ersetzt wird (17). Weiter zeigt sie, dass für monadische Formeln zweiter Stufe mit freien Variablen zweiter Stufe ein *Histogramm* berechnet werden kann, welches für die möglichen Kardinalitäten der freien Variablen angibt, wie viele erfüllende Belegungen existieren. Durch das Histogramm kann für viele Probleme nicht nur berechnet werden, ob eine Lösung existiert, sondern auch, wie viele Lösungen existieren. In dieser Arbeit wird ein Framework zur Anwendung der genannten Ergebnisse erstellt und verschiedene Probleme unter Verwendung dieses Frameworks untersucht.

Das Framework besteht aus einer Erweiterung der *parametrisierten Komplexitätstheorie*, in der komplexitätstheoretische Untersuchungen unter besonderer Beachtung von Problemparametern durchgeführt werden. Die Erweiterung ergänzt die parametrisierte Komplexitätstheorie, die sich bisher an der Untersuchung von Zeitklassen orientiert hat, um Platzklassen und einen passenden Reduktionsbegriff. Anhand dieser Platzklassen wird anschließend eine erste systematische Untersuchung von parametrisierten Problemen bezüglich ihrer platzeffizienten Berechnung durchgeführt. Dabei werden verschiedene Techniken zur Anwendung der Sätze von Bodlaender und Courcelle in ihren Logspace-Versionen vorgestellt.

Eine Untersuchung von Problemen bezüglich ihrer platzeffizienten Berechenbarkeit ist von großer praktischer Relevanz. Eines der heutzutage meistverwendeten Programmierparadigmen ist das der *dynamischen Programmierung* (4). Dieses Paradigma wird zur Berechnung von Optimierungsproblemem unter Beachtung des *Optimalitätsprinzips von Bellman* verwendet. Das Optimalitätsprinzip sagt aus, dass sich bei einigen Optimierungsproblemen jede optimale Lösung aus optimalen Teillösungen des Problems ergibt. Das Paradigma der dynamischen Programmierung greift dieses Prinzip auf, indem ein Problem in viele kleinere Teilprobleme zerlegt wird und für diese optimale Teillösungen berechnet werden. Aus diesen Teillösungen werden iterativ optimale Lösungen für größere Teilprobleme und schließlich das Gesamtproblem berechnet. Hierbei müssen bereits ermittelte Teillösungen gespeichert werden, da auf diese beständig zurückgegriffen wird. Konzeptionell lässt sich ein Algorithmus, der nach dem Paradigma der dynamischen Programmierung arbeitet, als eine Kombination aus einer Tabelle und einem Algorithmus betrachten, bei dem der Algorithmus die Tabelleneinträge berechnet.

Ein klassisches Problem, welches mittels dynamischer Programmierung berechnet werden kann, ist SUBSET-SUM. Hierbei soll für eine gegebene Menge von natürlichen Zahlen $S = \{w_1, w_2, \ldots, w_n\}$ ermittelt werden, ob eine Teilmenge $S' \subseteq S$

existiert, sodass die Summe ihrer Elemente einem gegebenen Zielwert $w$ entsprechen. Ein einfacher Algorithmus zur Berechnung von SUBSET-SUM benötigt Zeit $O(nw)$ und Speicherplatz $O(w)$ (37). Da $w$ binär kodiert ist, handelt es sich hierbei um einen Exponentialzeitalgorithmus mit exponentiellem Platzverbrauch. Ähnliche Algorithmen existieren für Probleme wie KNAPSACK, BIN-PACKING oder TRAVELLING-SALESMAN. Solche Algorithmen sind oftmals nicht praxistauglich, da der exponentiell große Speicherplatzbedarf nicht zur Verfügung steht. Es ist daher wichtig, platzeffizientere Algorithmen zu finden, falls solche existieren. Erste Untersuchungen (31) befassen sich bereits mit diesem Problem; eine systematische Untersuchung wird mit dieser Arbeit begonnen.

Kapitel 2 beinhaltet die für die Arbeit benötigten Grundlagen aus der Logik und der Graphentheorie. In Kapitel 3 wird nach einer Einführung in die parametrisierte Komplexitätstheorie das zur anschließenden Untersuchung von parametrisierten Problemen benötigte Framework vorgestellt. Kapitel 4 enthält systematische Untersuchungen von unterschiedlichen Problemen, die anhand des Frameworks durchgeführt wurden. Kapitel 5 gibt eine Zusammenfassung und einen Ausblick für weitere Untersuchungen bezüglich der parametrisierten Platzkomplexität.

# KAPITEL 2

# DIE SÄTZE VON BODLAENDER UND COURCELLE UND IHRE VARIANTEN

Dieses Kapitel enthält die logischen, graph- und komplexitätstheoretischen Grundlagen für diese Arbeit. Die logischen Grundlagen enthalten eine Einführung in die Logiken erster und zweiter Stufe sowie die monadische Logik zweiter Stufe. Der anschließende Abschnitt umfasst die graphtheoretischen Grundlagen, definiert insbesondere die zentralen Begriffe der Baumzerlegung und -weite und führt in diesem Zusammenhang einige wichtige Klassen von Graphen auf. Das Kapitel schließt mit den Sätzen von Bodlaender und Courcelle in den von Elberfeld, Jakoby und Tantau formulierten Logspace- und Kardinalitäts-Versionen.

Die Ausführungen zu den logischen Grundlagen orientieren sich an den ersten Kapiteln des Buches von Immerman (24). Für eine tiefergehende Einführung in die Logik eignen sich die entsprechenden Kapitel aus den Büchern von Papadimitriou (37) und Flum und Grohe (18). Einführungen in die Graphentheorie bieten Harary (21) und Diestel (15) an. Einführungen in die Komplexitätstheorie finden sich bei Reischuk (38) und Papadimitriou (37).

## 2.1 MONADISCHE LOGIK ZWEITER STUFE

Logik, so formuliert es Immerman (24), ist die mathematische Modellierung der Mathematik. Immerman zeigt, dass die Komplexität logischer Beschreibungen von Problemen der Informatik mit der Berechnungskomplexität dieser Probleme korrespondiert. In dieser Arbeit werden logische Beschreibungen der monadischen Logik zweiter Stufe verwendet, um auf die Platzkomplexität von Problemen schließen zu können. Die monadische Logik zweiter Stufe wird daher in diesem Abschnitt eingeführt.

Um über Sachverhalte, die durch logische Strukturen modelliert werden, Aussagen machen zu können, werden zunächst Bezeichner benötigt, denen anschließend eine Bedeutung gegeben werden kann. Diese Bezeichner werden in der *logischen Signatur* zusammengefasst.

DEFINITION 2.1. Eine *logische Signatur* $\tau$ besteht aus einer Menge von *Relations-* und *Konstantensymbolen* und einer *Stelligkeitsfunktion*, die jedem Relationssymbol eine echt positive natürliche Zahl, ihre *Stelligkeit*, zuordnet. ■

Typischerweise sind die Relationssymbole lateinische Großbuchstaben und die Konstantensymbole lateinische Kleinbuchstaben. In dieser Arbeit werden die Stelligkeiten als Exponenten des Symbols notiert, falls diese nicht klar aus dem

Kontext ersichtlich sind. Konstantensymbole erhalten dabei den Exponenten 0. Um im Folgenden zu notieren, dass ein Relationssymbol $R$ ein Element einer Signatur $\tau$ ist, wird $R \in \tau$ geschrieben. Die Symbole einer Signatur $\tau$ werden in Form einer Liste notiert. Ein Beispiel eine Signatur ist $\tau = (E^2, s^0, t^0)$ mit dem zweistelligen Relationssymbol $E$ und den beiden Konstanten $s$ und $t$. Die Symbole einer solchen Signatur haben so noch keine Bedeutung. Eine Bedeutung erhalten sie durch eine zu der Signatur passende *logische Struktur*.

■ DEFINITION 2.2. Für eine gegebene logische Signatur $\tau$ ist eine *logische Struktur*, eine sogenannte $\tau$-*Struktur*, ein Tupel $\mathcal{S}$ aus

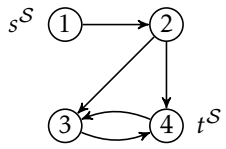
1. einer nichtleeren Menge $U$, dem *Universum*,
2. einer *Relation* $R^{\mathcal{S}} \in U^r$ für jedes Relationssymbol $R \in \tau$ mit der Stelligkeit $r \geq 1$ und
3. jeweils einem Element $c^{\mathcal{S}}$ des Universums, einer *Konstanten*, für jedes Konstantensymbol $c \in \tau$.

Die Menge der $\tau$-Strukturen wird mit $\mathrm{Struc}(\tau)$ bezeichnet. ■

Für die obige Beispielsignatur $\tau = (E^2, s^0, t^0)$ ist eine mögliche Beispielstruktur $\mathcal{S} \in \mathrm{Struc}(\tau)$

$$
\begin{aligned}
\mathcal{S} &= (V, E^{\mathcal{S}}, s^{\mathcal{S}}, t^{\mathcal{S}}),\\
V &= \{1,2,3,4\},\\
E^{\mathcal{S}} &= \{(1,2),(2,3),(2,4),(3,4),(4,3)\},\\
s^{\mathcal{S}} &= 1,\\
t^{\mathcal{S}} &= 4.
\end{aligned}
$$

Diese Struktur lässt sich als Graph darstellen, wie in Abbildung 2.1 gezeigt.




■ ABBILDUNG 2.1. Die Grafik zeigt die Darstellung der logischen Struktur $\mathcal{S}$ durch Knoten und Kanten. Ein Element $u$ des Universums wird als Knoten $(u)$, die Relation $E^{\mathcal{S}}$ für Elemente $(u,v) \in E^{\mathcal{S}}$ durch Kanten $(u) \to (v)$ dargestellt. Die Konstanten $s^{\mathcal{S}}$ und $t^{\mathcal{S}}$ sind beschriftet.

Durch logische Signaturen und Strukturen können Sachverhalte verschiedenster Art modelliert werden. Interessant werden diese Modelle jedoch erst dann, wenn Aussagen über diese formuliert und auf ihre Gültigkeit hin überprüft werden können. Die Formulierung solcher Aussagen geschieht durch *logische Formeln*, die nun eingeführt werden. Zunächst werden logische Formeln erster Stufe, darauf

aufbauend die logischen Formeln zweiter Stufe und als Spezialfall die monadischen logischen Formeln zweiter Stufe definiert. Anschließend wird die Gültigkeit von logischen Strukturen bezüglich dieser Formeln betrachtet.

■ DEFINITION 2.3. Die Klasse der logischen Formeln erster Stufe mit Gleichheit, der FO-*Formeln*, ist für eine Signatur $\tau$ induktiv definiert. Hierbei sind *Variablensymbole erster Stufe* Symbole, in der Regel lateinische Kleinbuchstaben, die noch nicht in der Signatur verwendet worden sind. Sind $t_1, \ldots, t_r$ Konstantensymbole oder Variablensymbole erster Stufe und ist $R \in \tau$ ein Relationssymbol mit der Stelligkeit $r$, so ist $R(t_1, \ldots, t_r)$ eine logische Formel erster Stufe. Sind $a$ und $b$ Konstanten- oder Variablensymbole erster Stufe, so ist $(a = b)$ eine logische Formel erster Stufe. Sind $\phi$ und $\psi$ zwei logische Formeln erster Stufe und ist $x$ ein Variablensymbol erster Stufe, so sind $(\neg\phi)$, $(\phi \wedge \psi)$ und $(\exists x.\phi)$ ebenfalls logische Formeln erster Stufe.

Ist $(\exists x.\phi)$ eine logische Formel erster Stufe, so heißt der Teil von $\phi$, der verbleibt, wenn alle Teilformeln der Form $(\exists x.\phi')$ aus $\phi$ entfernt wurden, der *Bindungsbereich* der Variable $x$ bezüglich des Quantors $\exists$. Diese Variable heißt dann auch *gebunden*. Eine nicht gebundene Variable heißt *frei*.

Die Operatoren haben unterschiedliche *Bindungsstärken* und lauten in abnehmender Reihenfolge nach der Bindungsstärke sortiert $\neg$, $\wedge$, $\exists$. ■

Die Bindungsstärke dient der übersichtlicheren Darstellung von Formeln. So können Klammern weggelassen werden, wenn dadurch die Eindeutigkeit einer logischen Formel nicht beeinträchtigt wird. Beispielsweise gilt $\varphi = (\exists a.((a \wedge b) \vee (c \wedge d))) \wedge a \equiv (\exists a.a \wedge b \vee c \wedge d) \wedge a$.

■ DEFINITION 2.4. Die induktive Definition der Klasse der logischen Formeln zweiter Stufe, der SO-*Formeln*, entspricht der Definition der Klasse der logischen Formeln erster Stufe mit folgenden Ergänzungen: Sind $t_1, \ldots, t_n$ Variablen erster Stufe oder Konstantensymbole, so ist $R(t_1, \ldots, t_n)$ eine logische Formel zweiter Stufe, wenn das Relationssymbol $R$ die Stelligkeit $n$ hat und nicht in der Signatur vorkommt. Das Symbol $R$ wird dann auch als *Variable zweiter Stufe* bezeichnet. Ist $R$ eine Variable zweiter Stufe und $\phi$ eine logische Formel zweiter Stufe, so ist auch $\exists R.\phi$ eine logische Formel zweiter Stufe. ■

■ DEFINITION 2.5. Die Klasse der monadischen logischen Formeln zweiter Stufe, der MSO-*Formeln*, ist die Klasse der logischen Formeln zweiter Stufe, deren Variablen zweiter Stufe die Stelligkeit 1 haben. ■

Sind $v_1, \ldots, v_k$ die freien Variablen einer logischen Formel $\phi$, so wird dies auch mit $\phi(v_1, \ldots, v_k)$ notiert.

Abhängig von der betrachteten logischen Struktur sind Aussagen, die durch logische Formeln und die zugehörige Signatur gebildet worden sind, gültig oder nicht gültig.

■ DEFINITION 2.6. Eine *Belegung* einer gegebenen Struktur ist eine Abbildung $\alpha$, die jeder freien Variablen erster Stufe ein Element aus dem Universum und jeder freien Variablen zweiter Stufe mit Stelligkeit $n$ eine $n$-stellige Relation über dem Universum zuweist. Aus technischen Gründen ist in dieser Arbeit die Abbildung $\alpha$ weiter so definiert, dass $\alpha(c) = c^{\mathcal{S}}$ für eine Konstante $c^{\mathcal{S}}$ und $\alpha(R) = R^{\mathcal{S}}$ für eine Relation $R^{\mathcal{S}}$ der betrachteten Struktur $\mathcal{S}$ gilt. ■

■ DEFINITION 2.7. Für eine Struktur $\mathcal{S}$, eine Belegung $\alpha$ und eine Formel $\phi$ ist die Struktur $\mathcal{S}$ ein *Modell* von $\phi$, wenn $\phi$ mit der Belegung $\alpha$ wahr ist. Dies wird durch die Schreibweise $(\mathcal{S}, \alpha) \vDash \phi$ abgekürzt. Ist die Formel $\phi$ unabhängig für alle Belegungen unter der Struktur $\mathcal{S}$ wahr, so wird dies durch $\mathcal{S} \vDash \phi$ abgekürzt. ■

■ DEFINITION 2.8. Es seien $a$ und $b$ Variablen erster Stufe oder Konstantensymbole, $\phi$ und $\psi$ logische Formeln, $R$ ein Relationssymbol oder eine Variable zweiter Stufe mit Stelligkeit $n$. Der *Wahrheitswert* für eine gegebene Struktur $\mathcal{S}$ und eine zugehörige Belegung ist induktiv definiert:

$$
\begin{aligned}
(\mathcal{S}, \alpha) &\vDash (a = b) \text{ gdw. } \alpha(a) = \alpha(b), \\
(\mathcal{S}, \alpha) &\vDash R(a_1, \ldots, a_n) \text{ gdw. } (\alpha(a_1), \ldots, \alpha(a_n)) \in \alpha(R), \\
(\mathcal{S}, \alpha) &\vDash (\neg\phi) \text{ gdw. nicht } (\mathcal{S}, \alpha) \vDash \phi, \\
(\mathcal{S}, \alpha) &\vDash (\phi \wedge \psi) \text{ gdw. } (\mathcal{S}, \alpha) \vDash \phi \text{ und } (\mathcal{S}, \alpha) \vDash \psi, \\
(\mathcal{S}, \alpha) &\vDash (\exists x.\phi) \text{ gdw. ein } a \in U \text{ existiert, sodass } (\mathcal{S}, \alpha[x \leftarrow a]) \vDash \phi \\
&\text{mit } \alpha[x \leftarrow a](y) = \begin{cases} \alpha(y), & \text{falls } y \neq x \\ a, & \text{falls } y = x. \end{cases}
\end{aligned}
$$

■

Um die Lesbarkeit der logischen Formeln zu verbessern, gelten die üblichen Abkürzungen, die im Folgenden noch einmal dargestellt sind.

■ DEFINITION 2.9. Es gelten die folgenden Abkürzungen:

$$
\begin{aligned}
a \neq b &\text{ steht für } \neg(a = b), \\
\exists xy.\phi &\text{ steht für } \exists x.\exists y.\phi \\
\forall x.\phi &\text{ steht für } \neg\exists x.\neg\phi, \\
\phi \vee \psi &\text{ steht für } \neg(\neg\phi \wedge \neg\psi), \\
\phi \rightarrow \psi &\text{ steht für } \neg\phi \vee \psi, \\
\phi \leftrightarrow \psi &\text{ steht für } (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi).
\end{aligned}
$$

Hierbei sind $a$ und $b$ Variablensymbole erster Stufe oder Konstantensymbole, $x$ und $y$ Variablensymbole erster oder zweiter Stufe und $\phi$ und $\psi$ logische Formeln zweiter Stufe. ■

Mit dieser Definition sind die logischen Grundlagen für diese Arbeit gelegt. Im Folgenden sind einige Beispiele zur Anwendung dieser Formeln gegeben, die noch-

mals die obige Beispielstruktur $\mathcal{S}$ verwenden, wie sie in Abbildung 2.1 dargetellt ist.

Eine logische Formel erster Stufe ist $\phi_{\text{NO-LOOPS}} = \forall x.\neg E(x,x)$, die genau dann in einer logischen Struktur wahr ist, wenn kein Knoten existiert, welcher eine Kante zu sich selbst besitzt, also kein Element des Univerums bezüglich der Relation $E^{\mathcal{S}}$ mit sich selbst in Relation steht. Es gilt $\mathcal{S} \vDash \phi_{\text{NO-LOOPS}}$. Eine weitere logische Formel erster Stufe ist $\phi_{\text{UNDIR}} = \forall xy.E(x,y) \to E(y,x)$, die genau dann wahr ist, wenn die Kantenrelation $E^{\mathcal{S}}$ symmetrisch ist. Dies ist in der Struktur $\mathcal{S}$ nicht der Fall; es gilt $\mathcal{S} \nvDash \phi_{\text{UNDIR}}$. Die Struktur $\mathcal{S}$ ist kein Modell dieser Formel.

Ein Beispiel für eine logische Formel zweiter Stufe ist $\phi_{\text{CLIQUE}}(C) = \forall uv.C(u) \wedge C(v) \wedge (u \neq v) \to E(u,v)$, die genau dann wahr ist, wenn die einstellige Relation $C$ eine Clique ist, also eine Menge von Knoten, von denen jeder mit jedem verbunden ist. Für die Beispielstruktur $\mathcal{S}$ gilt $(\mathcal{S},\alpha) \vDash \phi_{\text{CLIQUE}}$ mit $\alpha(C) = \{3,4\}$.

Viele Sachverhalte aus der Graphentheorie lassen sich durch MSO-Formeln ausdrücken. Für diese Arbeit ist der *reflexiv-transitive* Abschluss von besonderem Interesse, der im Folgenden definiert und dessen Ausdrückbarkeit mittels einer MSO-Formel bewiesen wird.

■ DEFINITION 2.10. Der *transitive Abschluss* $\varphi^+$ einer logischen Formel $\varphi$ mit zwei freien Variablen $u_\varphi$ und $v_\varphi$ ist eine logische Formel, die genau dann für zwei Elemente $u$ und $v$ wahr ist, wenn Elemente $w_1, w_2, \ldots, w_k$ mit $u = w_1$ und $v = w_k$ existieren, sodass für alle $i \in \{1,2,\ldots,k-1\}$ die Formel $\varphi(w_i, w_{i+1})$ gilt.

Der *reflexiv-transitive Abschluss* $\varphi^*$ ist dann definiert als

$$\varphi^*(u_\varphi, v_\varphi) = \varphi^+(u_\varphi, v_\varphi) \vee (u_\varphi = v_\varphi).$$ ■

Der reflexiv-transitive Abschluss ist nützlich, um die Existenz von Pfaden in einem Graphen auszudrücken, wie in Abschnitt 2.2 gezeigt wird, und kann weiter durch eine MSO-Formel ausgedrückt werden.

■ LEMMA 2.11. *Der reflexiv-transitive Abschluss $\varphi^*$ einer logischen Formel $\varphi$ mit zwei freien Variablen $u_\varphi$ und $v_\varphi$ kann durch eine* MSO*-Formel ausgedrückt werden.* ■

■ BEWEIS. Die folgende Formel $\phi_{\varphi\text{-TC}}(u,v)$ ist genau dann wahr, wenn $\varphi^*(u,v)$ gilt:

$$\phi_{\varphi\text{-TC}}(u,v) = \forall X.\phi_{\varphi\text{-CLOSED}}(X) \wedge X(u) \to X(v) \text{ mit}$$
$$\phi_{\varphi\text{-CLOSED}}(X) = \forall uv.\varphi(u,v) \wedge X(u) \to X(v).$$

Die Formel $\phi_{\varphi\text{-CLOSED}}(X)$ beschreibt, ob eine einstellige Relation $X$ unter der logischen Formel $\varphi$ abgeschlossen ist, also für jedes Element $u$ aus $X$ auch all die Elemente $v$ in $X$ sind, für die $\varphi(u,v)$ gilt.

Um die Korrektheit der Formel $\phi_{\varphi\text{-TC}}(u,v)$ zu zeigen, seien $u$ und $v$ zunächst zwei Elemente, sodass eine Elementfolge $w_1, w_2, \ldots, w_k$ mit $u = w_1$ und $w_k = v$ existiert,

für die $\varphi(w_i, w_{i+1})$ mit $i \in \{1, 2, \ldots, k-1\}$ gilt. Dann gilt für alle abgeschlossenen Mengen mit $u$, dass sie auch die Elemente $\{w_2, \ldots, w_k\}$ und damit insbesondere $v$ enthalten.

Sind $u$ und $v$ nun zwei Elemente, sodass keine wie oben betrachtete Elementfolge existiert, dann existiert eine abgeschlossene Elementmenge, die zwar $u$, aber nicht $v$ enthält. Es gilt dann also nicht für alle abgeschlossenen Mengen, die $u$ enthalten, dass sie auch $v$ enthalten. ■

## 2.2 BESCHRÄNKTE BAUMWEITE

Für viele NP-vollständige Graphprobleme lassen sich Algorithmen mit polynomieller Zeit- oder logarithmischer Platzkomplexität finden, wenn die Klasse der betrachteten Graphen auf Bäume oder auf seriell-parallele Graphen eingeschränkt wird (41, 26, 27). Es zeigte sich, dass dies auch für Graphen gilt, die eine gewisse Ähnlichkeit zu Bäumen haben. Ein Maß für die Ähnlichkeit eines Graphen mit einem Baum definierten Robertson und Seymour (39) mit den Begriffen der *Baumzerlegung* und der *Baumweite*. Für Graph-Klassen, deren Baumweite einer Konstanten entspricht, die also eine Art konstanter Ähnlichkeit zu Bäumen haben, lassen sich Baumzerlegungen effizient berechnen (6, 17) und Probleme unter Verwendung dieser Baumzerlegungen effizient lösen (3, 5). In diesem Abschnitt werden, aufbauend auf der oben eingeführen Logik, Graphen und anschließend die von Robertson und Seymour eingeführten Begriffe der Baumzerlegung und -weite definiert.

■ DEFINITION 2.12. Ein *Graph* ist eine logische Struktur $\mathcal{S}$ mit einer zweistelligen Relation $E^{\mathcal{S}}$, der *Kantenrelation*. Er ist *ungerichtet*, wenn seine Kantenrelation symmetrisch ist, andernfalls ist er *gerichtet*. Die Elemente des Universums heißen *Knoten*. Stehen zwei Knoten $u$ und $v$ in Relation zueinander, gilt also $E(u, v)$, so sagt man auch, dass eine *Kante* von $u$ nach $v$ in dem Graphen existiert. ■

Die in Abschnitt 2.1 betrachtete Beispielsignatur $\tau = (E^2, s, t)$ entspricht der eines Graphen mit zwei ausgezeichneten Knoten $s^{\mathcal{S}}$ und $t^{\mathcal{S}}$. Eine Darstellung eines Graphen ist in Abbildung 2.1 auf Seite 6 zu sehen.

■ DEFINITION 2.13. Ein *Walk* in einem Graphen ist eine Folge von Knoten, sodass für jeden bis auf den letzten Knoten der Folge gilt, dass er mit seinem Nachfolgerknoten in Relation steht. Ein *Pfad* ist ein Weg, in dem jeder Knoten nur einmal vorkommt. ■

■ DEFINITION 2.14. Ein Graph ist ein *Baum*, wenn ein Knoten $r$ existiert, sodass von jedem Knoten des Graphen genau ein Pfad zu $r$ führt. Dieser Knoten heißt die *Wurzel* des Baumes. Die *Kinder* eines Knotens $u$ sind dann die Knoten, die über eine direkte Kante mit $u$ verbunden sind und deren Pfad zur Wurzel über $u$ führt. ■

Jeder Graph hat eine mehr oder weniger große Ähnlichkeit mit einem Baum, wenn mehrere Knoten des Graphen zu einem Knoten zusammengefasst werden. Beispielhaft ist dies in Abbildung 2.2 dargestellt.

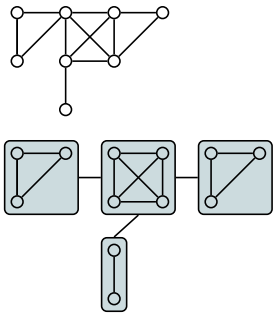


ABBILDUNG 2.2. Durch Zusammenfassen mehrerer Knoten zu einem neuen Knoten entsteht aus dem obigen Graphen ein Baum. Bei dem dargestellten Baum handelt es sich um eine Baumzerlegung des obigen Graphen mit Weite 3.

Wie baumähnlich ein Graph ist, wird durch die *Baumweite* ausgedrückt, die sich aus der *Baumzerlegung* des betrachteten Graphen ergibt.

DEFINITION 2.15. Eine *Baumzerlegung* eines Graphen $\mathcal{G}$ ist ein Tupel aus einem Baum $\mathcal{T}$ und einer bijektiven Abbildung, die jedem Knoten von $\mathcal{T}$ eine einstellige Relation $B_i$ in $\mathcal{G}$, einen *Bag*, mit folgenden Eigenschaften zuordnet:

1. Jeder Knoten von $\mathcal{G}$ befindet sich in mindestens einem Bag.
2. Für jede Kante von $\mathcal{G}$ existiert mindestens ein Bag, sodass ihre beiden Endknoten in diesem Bag sind.
3. Für jeden Knoten $a$ von $\mathcal{G}$ ist der Teilbaum von $\mathcal{T}$ zusammenhängend, der verbleibt, wenn alle Knoten und zugehörigen Kanten von $\mathcal{T}$ entfernt werden, deren Bags $a$ nicht enthalten.

Die *Weite* einer Baumzerlegung ist die maximale Anzahl von Knoten in einem der Bags minus 1.

Abbildung 2.2 zeigt einen Graphen mit einer zugehörigen Baumzerlegung.

Für einen Graphen gibt es in der Regel mehrere Baumzerlegungen. Mittels der Baumzerlegungen eines Graphen kann bewertet werden, wie ähnlich der Graph einem Baum ist. Hierfür wird der Begriff der *Baumweite* verwendet.

DEFINITION 2.16. Für einen Graphen $\mathcal{G}$ ist die *Baumweite* die minimale Weite über allen Baumzerlegungen. Eine Familie von Graphen hat eine *beschränkte Baumweite*, wenn die Baumweiten aller Graphen der Familie durch eine Konstante nach oben beschränkt sind.

Die Baumweite beschreibt die Ähnlichkeit eines Graphen zu einem Baum. Bäume haben eine sehr hohe Ähnlichkeit zu Bäumen.

LEMMA 2.17. *Bäume sind gerade die Graphen mit Baumweite* 1.

Dieses Lemma kann per Induktion über die Anzahl der Knoten eines Baumes bewiesen werden.

Viele Eigenschaften von Graphen sind durch monadische Formeln erster Ordnung ausdrückbar. Ein Beispiel dafür sind Walks.

LEMMA 2.18. *Es ist durch eine* MSO*-Formel* $\phi_{\varphi\text{-PATH}}(P,s,t)$ *ausdrückbar, dass eine einstellige Relation* $P$ *die Knoten eines Pfades von* $s$ *nach* $t$ *via der logischen Formel* $\varphi$ *beinhaltet.*

BEWEIS. Eine mögliche MSO-Formel ist

$$\phi_{\varphi\text{-PATH}}(P,s,t) = P(s) \wedge P(t) \wedge \phi_{\varphi\text{-TC}}(s,t) \text{ mit}$$
$$\varphi(u_\varphi, v_\varphi) = E(u_\varphi, v_\varphi) \wedge P(u_\varphi) \wedge P(v_\varphi).$$

## 2.3 PLATZKOMPLEXITÄT

In diesem Abschnitt werden die für die nächsten Kapitel benötigten komplexitätstheoretischen Grundlagen gelegt. Diese beinhalten die Kodierung von Ein- und Ausgaben für Turingmaschinen sowie die Formalisierung von Berechnungen. Es werden die benötigten Komplexitätsklassen und eine für diese Klassen geeignete Reduktion definiert. Eine tiefergehende Einführung in die Komplexitätstheorie bieten die Bücher von Papadimitriou (37), Reischuk (38) und Flum und Grohe (18).

Die Komplexitätstheorie befasst sich mit der algorithmischen Komplexität von Problemen, die sich im Ressourcenverbrauch von Rechnermodellen äußert, die diese Probleme lösen. Die zwei in dieser Arbeit zentralen Ressourcen sind dabei der Zeit- und der Speicherplatzbedarf eines zur Lösung eines Problems verwendeten Maschinenmodells. Der Verbrauch dieser Ressourcen wird dabei sinnvollerweise in Abhängigkeit von der Größe der Eingabe angegeben, da eine Maschine ein Problem für eine kleine Eingabe schneller lösen wird als für eine große. Um den Größenbegriff für eine Eingabe definieren zu können, ist zunächst die Betrachtung von Eingabekodierungen nötig.

Ein- und Ausgaben von Turingmaschinen sind logische Strukturen, die als binäre Zeichenketten kodiert werden. Um beliebige logische Strukturen als binäre Zeichenkette kodieren zu können, wird im Folgenden zunächst die Darstellung dieser Zeichenketten als logische Struktur und anschließend die Abbildung von beliebigen logischen Strukturen auf Strukturen von Zeichenketten und umgekehrt betrachtet. Mit diesen Werkzeugen lassen sich dann logische Strukturen von Turingmaschinen lesen, bearbeiten und ausgeben.

Um binäre Zeichenketten zu kodieren, wird die logische Signatur $\tau_s = (\leq^2, I_1^1)$ verwendet. Die Zeichenkette 0011 entspricht dann der logischen $\tau_s$-Struktur $\mathcal{S} =$

$(U, \leq^{\mathcal{S}}, I_1^{\mathcal{S}})$ mit

$$U = \{0,1,2,3\},$$
$$I_1^{\mathcal{S}} = \{2,3\}$$

und der Relation $\leq^{\mathcal{S}}$, die der üblichen Ordnung über dem Universum entspricht. Die Größe des Universums gibt die Länge der Zeichenkette und $I_1^{\mathcal{S}}$ diejenigen Indizes an, an denen in der Zeichenkette eine 1 steht. Zeichenketten über größeren Alphabeten können entsprechend kodiert werden, indem für jedes Symbol $\sigma$ des betrachteten Alphabets eine Relation $I_\sigma^{\mathcal{S}}$ die Indizes beinhaltet, an denen in der Zeichenkette ein $\sigma$ vorkommt.

Um auch logische Strukturen mit anderen Signaturen betrachten zu können, wird das Kodierungsschema $\text{bin}_\tau$ festgelegt, welches eine logische Struktur über der Signatur $\tau$ auf eine logische Struktur mit der Signatur $\tau_s$ abbildet.

DEFINITION 2.19. Für eine gegebene Signatur $\tau = (R_1^{r_1}, \ldots, R_k^{r_k}, c_1, \ldots, c_l)$ und eine gegebene logische $\tau$-Struktur $\mathcal{S} = (\{0, \ldots, n-1\}, R_1^{\mathcal{S}}, \ldots, R_k^{\mathcal{S}}, c_1^{\mathcal{S}}, \ldots, c_l^{\mathcal{S}})$ mit der Ordnungsrelation $\leq^{\mathcal{S}}$ ist die Abbildung $\text{bin}_\tau$ wie folgend definiert:

1. Eine Relation $R_i^{\mathcal{S}}$ wird als binärer String der Länge $n^{r_i}$ kodiert, wobei eine 1 an der Stelle $j$ angibt, dass das Tupel $(x_1, \ldots, x_{r_i})$ mit $\sum_{h=1}^{r_i} x_h \cdot n^{h-1} = j$ in $R_i^{\mathcal{S}}$ ist. In der Summe wird mit $x_h$ die Nummer des Elements $x_h$ bezeichnet.
2. Eine Konstante $c_i^{\mathcal{S}}$ wird als Binärzahl der Länge $\lceil \log n \rceil$ kodiert, die ihre Nummer angibt.
3. Die Binärkodierung der Struktur ist dann die Konkatenation der Binärkodierungen der Relationen und Konstanten entsprechend ihrer Vorkommnisse in der Signatur. ■

Für binäre Strings handelt es sich bei dieser Abbildung um die identische Abbildung, Graphen werden auf ihre Adjazenzmatrizen abgebildet.

Diese Kodierung erfüllt zwei sinnvolle Eigenschaften: Sie ist platzeffizient, da es sich um eine binäre Kodierung handelt, und sie ist leicht berechenbar, da sie und ihre Umkehrfunktion FL-berechenbar sind. Eine genauere Ausführung zu FL-Berechenbarkeit findet sich später in diesem Abschnitt, nach der Einführung von Komplexitätsklassen.

Die Größe einer Eingabe lässt sich dann als die Länge der Zeichenkette betrachten, die die Eingabe kodiert. Dies entspricht der Größe des Universums der kodierten logischen Struktur. Für eine Zeichenkette $x$ bezeichne $|x|$ deren Länge. Durch die binäre Kodierung kann die Länge des kodierenden Binärstrings von der Größe des eigentlich betrachteten Universums abweichen, jedoch ist die Länge des Binärstrings polynomiell durch die Größe des Universums beschränkt, was für die Betrachtungen in dieser Arbeit ausreichend ist.

■ DEFINITION 2.20. Ein *Entscheidungsproblem* $L$ ist eine Menge von Strukturen einer Signatur $\tau$. Ein *Funktionsproblem* ist eine Abbildung $I : \mathrm{Struc}(\tau) \to \mathrm{Struc}(\sigma)$ von Strukturen einer Signatur $\tau$ auf Strukturen einer Signatur $\sigma$.

Eine Turingmaschine *berechnet ein Entscheidungsproblem* $L$, wenn sie die Kodierung einer Eingabestruktur $\mathcal{S}$ genau dann akzeptiert, wenn $\mathcal{S} \in L$ gilt. Eine Turingmaschine *berechnet ein Funktionsproblem* $I$ genau dann, wenn nach dem Halt der Maschine für eine gegebene Kodierung einer Eingabestruktur $\mathcal{S}$ auf dem Ausgabeband die Kodierung der Struktur $I(\mathcal{S})$ steht. ■

Entscheidungs- und Funktionsprobleme können nach ihrer Berechnungskomplexität klassifiziert werden. Probleme mit gleicher Komplexität werden in Komplexitätsklassen zusammengefasst. Eine *Komplexitätsklasse* ist eine Klasse von Entscheidungs- oder Funktionsproblemen, für deren Berechnung definierte Ressourcenschranken eingehalten werden können. Diese Ressourcenschranken können beispielsweise der Zeit- oder Platzverbrauch von Turingmaschinen sein, die die Probleme berechnen, oder aber auch Klassen von logischen Formeln, mit deren Hilfe Aussagen über eine gegebene Struktur formuliert und deren Gültigkeit bezüglich der gegebenen Struktur zur Berechnung des Problems überprüft werden müssen. Übliche Komplexitätsklassen, die in dieser Arbeit betrachtet werden, sind folgende:

- L, die Klasse der Entscheidungsprobleme, die von Turingmaschinen berechnet werden können, deren Platzverbrauch logarithmisch in der Eingabelänge ist.
- FL, die Klasse der Funktionsprobleme, die von Turingmaschinen berechnet werden können, deren Platzverbrauch logarithmisch in der Eingabelänge ist.
- P, die Klasse der Entscheidungsprobleme, die von Turingmaschinen berechnet werden können, deren Zeitverbrauch polynomiell in der Eingabelänge ist.
- FP, die Klasse der Funktionsprobleme, die von Turingmaschinen berechnet werden können, deren Zeitverbrauch polynomiell in der Eingabelänge ist.
- NP, die Klasse der Entscheidungsprobleme, die von nichtdeterministischen Turingmaschine berechnet werden können, deren Zeitverbrauch polynomiell in der Eingabelänge ist.
- EXP, die Klasse der Funktionsprobleme, die von Turingmaschinen berechnet werden können, deren Zeitverbrauch exponentiell in der Eingabelänge ist.

Es ergibt sich bei Betrachtung der Entscheidungsprobleme die bekannte Inklusionskette $\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{EXP}$ (24, 38, 37).

Um ein Problem bezüglich seiner Komplexität zu klassifizieren, kann es mit anderen Problemen verglichen werden, von denen die Zugehörigkeiten zu Komplexitätsklassen bekannt sind. Hierfür werden *Reduktionen* verwendet, die eine Eingabe für ein gegebenes Problem $L_1$ in eine Eingabe für ein anderes Problem $L_2$

transformieren. So kann $L_1$ berechnet werden, indem für eine gegebene Eingabe zunächst die Reduktion und anschließend das Problem $L_2$ für die transformierte Eingabe berechnet wird. Im Folgenden wird der Reduktionsbegriff für die Klasse der Entscheidungsprobleme definiert, da dies für die Untersuchungen in dieser Arbeit ausreichend ist. Eine Diskussion von Reduktionsbegriffen für Funktionsprobleme findet sich im Buch von Papadimitriou (37).

■ DEFINITION 2.21. Es seien $L_1$ und $L_2$ zwei Entscheidungsprobleme. Eine *Many-One-Reduktion* ist dann eine Funktion $f$, sodass für alle Strukturen $\mathcal{S}$ über der Signatur von $L_1$ gilt $\mathcal{S} \in L_1$ gdw. $f(\mathcal{S}) \in L_2$. Das Problem $L_1$ heißt dann auf $L_2$ *many-one-reduzierbar*. ■

Durch die Reduktion eines Problems $L_1$ auf ein Problem $L_2$ ist es möglich, die Komplexität von $L_1$ abzuschätzen: Sie entspricht höchstens der Reduktionsfunktion auf der Eingabe für das Problem $L_1$ und der des Problems $L_2$ auf der Ausgabe der Reduktionsfunktion. Ist die Komplexität der Reduktionsfunktion geringer als die des Problems $L_2$, so lässt sich die Komplexität von $L_1$ durch die von $L_2$ abschätzen. Ein oft betrachteter Typ von Reduktionen sind FL-Reduktionen. Diese sind hinreichend schwach, um Probleme der Klassen P oder NP miteinander zu vergleichen. Ein einfaches Beispiel für eine FL-Reduktion ist die von $k$-CLIQUE auf das Problem $k$-INDEPENDENT-SET für ein fest gewähltes $k$. Die beiden Probleme sind wie folgend definiert:

$$k\text{-CLIQUE} = \{(V, E^{\mathcal{G}}) \mid \exists C . |C|_{\geq k} \wedge \forall uv . C(u) \wedge C(v) \rightarrow E(u,v)\},$$
$$k\text{-INDEPENDENT-SET} = \{(V, E^{\mathcal{G}}) \mid \exists S . |S|_{\geq k} \wedge \forall uv . S(u) \wedge S(v) \rightarrow \neg E(u,v)\},$$

wobei $|X|_{\geq k}$ eine logische Formel bezeichnet, die genau dann wahr ist, wenn $X$ eine Kardinalität von mindestens $k$ hat.

Es ist gut zu erkennen, dass die beiden Probleme große Ähnlichkeit haben: Ein Graph $\mathcal{G}$ hat genau dann eine $k$-Clique, wenn der Graph $\mathcal{G}'$, der aus den Knoten von $\mathcal{G}$ besteht und genau zwischen allen Paaren von Knoten eine Kante hat, die in $\mathcal{G}$ nicht durch eine Kante verbunden sind, eine unabhängige Menge der Größe $k$ besitzt. Diese Transformation von $\mathcal{G}$ auf $\mathcal{G}'$ lässt sich durch eine FL-Reduktion berechnen, indem eine Turingmaschine auf Eingabe einer Instanz von $k$-CLIQUE alle Bits der Adjazenzmatrix komplementiert und diese neue Struktur ausgibt.

Lässt sich ein Problem $L_1$ auf ein Problem $L_2$ reduzieren, so hat dies die intuitive Bedeutung, dass $L_1$ leichter zu berechnen ist als $L_2$, da $L_2$ für die Berechnung von $L_1$ benutzt werden kann. Hieraus ergeben sich die Begriffe der *Schwere* und der *Vollständigkeit* für eine Klasse.

■ DEFINITION 2.22. Ein Problem $L_1$ ist für eine Komplexitätsklasse $C$ *schwer* oder auch *hart*, wenn sich alle Probleme dieser Komplexitätsklasse auf $L_1$ reduzieren lassen. $L_1$ ist *vollständig* für $C$, wenn $L_1$ zusätzlich zur Schwere selbst in $C$ liegt. ■

Die für eine Komplexitätsklasse vollständigen Probleme sind als die schwersten Probleme dieser Klasse zu verstehen, da jedes andere Problem dieser Klasse sich auf dieses Problem reduzieren lässt, also intuitiv gesprochen leichter ist.

Die Klasse P wird aus mehreren Gründen als die Klasse der praktisch berechenbaren Probleme angesehen, während Probleme aus den höheren Klassen wie NP nicht als praktisch berechenbar eingestuft werden. Ein Grund hierfür ist, dass für viele Probleme aus P Algorithmen bekannt sind, deren Laufzeiten durch Polynome mit recht kleinen Exponenten beschränkt sind (19), während für viele Probleme der Klasse NP nur Algorithmen mit exponentieller Laufzeit existieren, deren Laufzeiten mit wachsenden Eingabegrößen inakzeptabel stark ansteigen. Jedoch sind es gerade die Probleme der Klasse NP, die von größerem Interesse sind, weshalb die Lösbarkeit dieser Probleme oft untersucht wird.

Eine Differenzierung von NP-vollständigen Problemen findet sich bei *Zahlenproblemen*. Für viele dieser Probleme existieren Algorithmen, deren Laufzeiten polynomiell von den numerischen Werten der in der Eingabe kodierten Zahlen abhängen. Solche Algorithmen werden als *Pseudo-Polynomialzeit-Algorithmen* bezeichnet. Nicht für alle Probleme existieren Pseudo-Polynomialzeit-Algorithmen, was zur Unterscheidung von *schwach-* und *stark*-NP-*vollständigen Problemen* führt.

DEFINITION 2.23. Ein Problem heißt *schwach*-NP-*vollständig*, wenn es NP-vollständig ist und ein Algorithmus für das Problem existiert, dessen Laufzeit polynomiell von der Länge der Eingabe und dem Wert der größten in der Eingabe kodierten Zahl abhängt. Existiert für ein NP-vollständiges Problem solch ein Algorithmus nicht, so heißt das Problem *stark*-NP-*vollständig*.

Die Laufzeit eines Pseudo-Polynomialzeit-Algorithmus bestimmt sich somit nicht nur durch die Länge der Eingabe, sondern auch durch einen Parameter der Eingabe, hier dem Wert der größten in der Eingabe kodierten Zahl.

Auch für andere Probleme lassen sich Algorithmen finden, deren Laufzeiten nicht ausschließlich von der Länge der Eingabe, sondern von Parametern der Eingabe abhängen. Eine Einführung in die *parametrisierte Komplexitätstheorie*, die sich mit der Untersuchung von Problemen unter Beachtung von Problemparametern beschäftigt, findet sich in Kapitel 3 dieser Arbeit.

## 2.4 DIE LOGSPACE-VERSIONEN DER SÄTZE VON BODLAENDER UND COURCELLE

Wie in Abschnitt 2.2 angesprochen, lassen sich viele NP-vollständige Graphprobleme in polynomieller Zeit lösen, wenn die betrachteten Probleme eine beschränkte Baumweite haben. Hierbei wird von der Baumzerlegung der Graphen Gebrauch gemacht, die zunächst berechnet werden muss. Bodlaender zeigte, dass die Berechnung einer Baumzerlegung mit minimaler Weite für Graphen von beschränkter Baumweite in linearer Zeit möglich ist (6). Unter Verwendung dieses Satzes zeigte

Courcelle, dass Aussagen, die in monadischer Logik zweiter Stufe formuliert sind, für Graphen von beschränkter Baumweite ebenfalls in linearer Zeit entschieden werden können (14). Diese Sätze stellen mächtige Werkzeuge zum Beweis der Existenz von Linearzeitalgorithmen für Probleme dar, die sich als Graphen mit beschränkter Baumweite modellieren und durch monadische Formeln zweiter Stufe ausdrücken lassen. Elberfeld, Jakoby und Tantau zeigten, dass die Sätze von Bodlaender und Courcelle ebenfalls gelten, wenn die lineare Zeitschranke durch eine logarithmische Platzschranke ersetzt wird (17). Diese Sätze werden in diesem Abschnitt betrachtet.

Das in Abschnitt 2.2 eingeführte Konzept der Baumweite lässt sich auf beliebige logische Strukturen erweitern.

■ DEFINITION 2.24. Baumzerlegungen von logischen Strukturen sind definiert wie Baumzerlegungen von Graphen mit folgenden Erweiterungen:

1. Jedes Element des Universums befindet sich in einem Bag.
2. Für jedes Relationssymbol $R^r$ und jedes Tupel $(a_1,\dots,a_r) \in R^{\mathcal{S}}$ existiert ein Bag, sodass die Elemente $a_1,\dots,a_r$ in diesem Bag sind.

Die Baumweite ist analog definiert. ■

Für Strukturen von beschränkter Baumweite lassen sich Baumzerlegungen platzeffizient berechnen.

■ SATZ 2.25: (17). *Für jedes $k \geq 1$ existiert eine logarithmisch platzbeschränkte Turingmaschine, die auf Eingabe einer logischen Struktur $\mathcal{S}$ mit maximaler Baumweite $k$ eine Baumzerlegung von $\mathcal{S}$ mit Baumweite $k$ ausgibt.* ■

Weiter zeigten Elberfeld, Jakoby und Tantau, dass für ein fest gewähltes $k$ entschieden werden kann, ob eine logische Struktur eine Baumzerlegung mit Weite $k$ existiert oder nicht.

■ SATZ 2.26: (17). *Für jedes $k \geq 1$ ist das Problem* TREEWIDTH-$k$, *welches genau die Strukturen enthält, deren Baumweite höchstens $k$ ist, auf logarithmischem Platz entscheidbar.* ■

Ist die Konstante $k$ nicht gegeben, so ist das Problem, zu entscheiden, ob eine Struktur höchstens Baumweite $k$ hat, NP-vollständig (2).

Der Satz von Courcelle wurde von Elberfeld, Jakoby und Tantau zu einer *Kardinalitäts-Version* verallgemeinert, die im Folgenden vorgestellt wird.

■ DEFINITION 2.27. Es sei $\varphi$ eine MSO-Formel mit den freien einstellige Variablen zweiter Stufe $X_1,\dots,X_d$ und $\mathcal{S}$ eine Struktur mit dem Universum $U$. Dann ist das *Lösungshistogramm* histogram$[\mathcal{S},\varphi]$ das $d$-dimensionale Feld natürlicher Zahlen, das am Index $(i_1,\dots,i_d) \in \{0,\dots,|U|\}^d$ als Wert die Anzahl der Teilmengen $R_1,\dots,R_d \subseteq U$ mit $|R_1| = i_1,\dots,|R_d| = i_d$ enthält, sodass $(\mathcal{S},\alpha) \vDash \varphi$ mit $\alpha(X_i) = R_i$ für $i \in \{1,\dots,d\}$ gilt. ■

Abbildung 2.3 zeigt ein Beispiel eines Lösungshistogramms für das Problem DOMINATING-SET und eine Beispielstruktur.

| $\lvert S\rvert$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $\text{histogram}[\mathcal{S}, \phi_{\text{DOMINATING-SET}}](\lvert S\rvert)$ | 0 | 3 | 3 | 1 |

■ ABBILDUNG 2.3. Für einen ungerichteten Graphen ist eine dominierende Menge eine Teilmenge der Knoten, sodass jeder Knoten des Graphen entweder in der Menge oder mit einem Knoten aus der Menge direkt verbunden ist. Dieses Problem lässt sich durch die MSO-Formel $\phi_{\text{DOMINATING-SET}}(S) \equiv \forall u . S(u) \vee \exists v . S(v) \wedge E(v,u)$ ausdrücken, wobei diese genau dann wahr ist, wenn $S$ eine dominierende Menge ist. Die obige Abbildung zeigt das Lösungshistogramm der Formel $\phi_{\text{DOMINATING-SET}}$ mit der freien Variablen $S$ für den dargestellten Graphen als Tabelle. Die erste Zeile der Tabelle gibt die Größe der Menge $S$, die zweite Zeile die Anzahl der Mengen mit der entsprechenden Kardinalität an.

Aus dem Lösungshistogramm können weitere Informationen gewonnen werden, wie beispielsweise die Anzahl der möglichen erfüllenden Belegungen einer monadischen Formel erster Stufe, die sich aus der Summe aller Einträge des Lösungshistogramms ergibt. Diese Informationen können für Strukturen mit beschränkter Baumweite auf logarithmischem Platz berechnet werden, wie der folgende Satz zeigt.

■ SATZ 2.28: (17). *Für alle $k \geq 1$ und jede* MSO*-Formel $\varphi(X_1, \ldots, X_d)$ existiert eine logarithmisch platzbeschränkte Turingmaschine, die auf Eingabe einer logischen Struktur $\mathcal{S}$ mit maximaler Baumweite $k$ das Lösungshistogramm* $\text{histogram}[\mathcal{S}, \varphi]$ *ausgibt.* ■

# KAPITEL 3

## PARAMETRISIERTE PLATZKOMPLEXITÄT

Der Zweig der *parametrisierten Komplexitätstheorie* beschäftigt sich mit der komplexitätstheoretischen Untersuchung von Problemen unter besonderer Beachtung von Parametern. In der klassischen Komplexitätstheorie wird der Ressourcenverbrauch für eine Berechnung in Abhängigkeit von der Eingabelänge gemessen, wodurch viele Eigenschaften der gegebenen Probleminstanz nicht beachtet werden, die entscheidenden Einfluss auf die Laufzeit eines Algorithmus haben können. In der parametrisierten Komplexitätstheorie wird die Laufzeit eines Algorithmus nicht nur in Abhängigkeit von der Eingabelänge gemessen, sondern auch in Abhängigkeit eines Parameters; eines numerischen Wertes, der sich aus der gegebenen Eingabe bestimmt. Dies erlaubt einen differenzierteren Blick auf die Komplexität eines Problems und ermöglicht den Entwurf von besseren Algorithmen für Umgebungen, in denen Informationen bezüglich der betrachteten Parameter gegeben sind (16, 18). Die hier verwendete Notation orientiert sich an der von Flum und Grohe (18).

DEFINITION 3.1. Sei $\tau$ eine Signatur. Eine *Parametrisierung* ist eine FL-berechenbare Abbildung $\kappa : \mathrm{Struc}(\tau) \to \mathbb{N}$, die einer Probleminstanz einen Parameter zurordnet. Ein *parametrisiertes Entscheidungsproblem* ist ein Tupel $(L, \kappa)$ aus einem Entscheidungsproblem $L \subseteq \mathrm{Struc}(\tau)$ und einer Parametrisierung $\kappa$. Ein *parametrisiertes Funktionsproblem* ist ein Tupel $(I, \kappa)$ aus einem Funktionsproblem $I$ und einer Parametrisierung $\kappa$.

Ein Beispiel für ein parametrisiertes Problem ist das folgende:

- $p$-VERTEX-COVER.
  - INSTANZ. Eine logische Struktur, bestehend aus einem Graphen $\mathcal{G}$ und einer natürlichen Zahl $k$.
  - PARAMETER. $k$.
  - FRAGE. Existiert eine Knotenüberdeckung für $\mathcal{G}$ der Größe $k$? Eine Knotenüberdeckung ist dabei eine Menge von Knoten, sodass jede Kante des Graphen zu mindestens einem der ausgewählten Knoten inzident ist.

Für parametrisierte Probleme lassen sich weitere Komplexitätsklassen definieren. Eine zentrale Komplexitätsklasse der parametrisierten Komplexitätstheorie ist die Klasse FPT der *Fixed-Parameter-Tractable-Problems*.

DEFINITION 3.2. Ein parametrisiertes Problem $(L, \kappa)$ ist in der Klasse FPT, wenn ein Algorithmus existiert, der $L$ entscheidet und dessen Laufzeit für eine Einga-

be $x$ durch $f(\kappa(x)) \cdot p(|x|)$ für eine berechenbare Funktion $f$ und ein Polynom $p$ beschränkt ist. Die korrespondierende Klasse der Funktionsprobleme sei FFPT. ■

Ein Problem ist in der Klasse FPT, wenn ein Algorithmus für das Problem existiert, dessen Laufzeit exponentiell oder möglicherweise noch stärker wachsend vom Parameter und polynomiell von der Eingabelänge abhängig ist. Ein Beispiel hierfür ist das oben genannte Problem $p$-VERTEX-COVER für das ein Algorithmus mit Laufzeit $O(1{,}2738^{\kappa(x)} + |x| \cdot \kappa(x))$ existiert. $p$-VERTEX-COVER ist also ein Problem der Klasse FPT (11).

Wie auch in der klassischen Komplexitätstheorie wird zur Untersuchung von Problemen und Problemklassen ein Reduktionsbegriff benötigt. In der parametrisierten Komplexitätstheorie wird typischerweise die FFPT-Reduktion verwendet.

■ DEFINITION 3.3. Es seien $(L_1, \kappa_1)$ und $(L_2, \kappa_2)$ zwei parametrisierte Probleme mit $L_1 \subseteq \text{Struc}(\tau_1)$ und $L_2 \subseteq \text{Struc}(\tau_2)$. Dann ist $f$ eine FFPT-*Reduktion* von $(L_1, \kappa_1)$ auf $(L_2, \kappa_2)$, wenn

1. für alle $x \in \text{Struc}(\tau_1)$ gilt, dass $x \in L_1$ gdw. $f(x) \in L_2$,
2. $(f, \kappa_1) \in \text{FFPT}$,
3. ein Polynom $g$ existiert, sodass $\kappa_2(f(x)) \leq g(\kappa_1(x))$ gilt. ■

Um die Klasse FPT herum rankt sich eine Vielfalt von weiteren Komplexitätsklassen. Eine dieser Klassen ist die parametrisierte Klasse XP, die zu EXP korrespondiert, der Klasse von Problemen, die sich durch Exponentialzeitalgorithmen entscheiden lassen. Die Klasse XP ist eine der wenigen Klassen, für die eine echte Teilmengenbeziehung zu FPT gezeigt werden konnte.

■ DEFINITION 3.4. Ein parametrisiertes Problem $(L, \kappa)$ ist in der Klasse XP, wenn für alle $k \geq 1$ gilt, dass $L_k = \{x \mid x \in L \text{ und } \kappa(x) = k\} \in \text{P}$ via eines festen Algorithmus. Die zu XP korrespondierende Klasse der Funktionsprobleme sei FXP. ■

Die Klasse XP lässt sich auch als Klasse der Probleme betrachten, die stückweise für konstante Parameter, in P liegen.

■ SATZ 3.5: (16, 18).

$$\text{FPT} \subset \text{XP}.$$

■

Die in Kapitel 2 betrachteten schwach-NP-vollständigen Probleme lassen sich parametrisieren und in der Klasse PPT, der *Pseudo-Polynomial-Time-Problems*, erfassen.

■ DEFINITION 3.6. Ein parametrisiertes Problem $(L, \kappa)$ ist in der Klasse PPT, wenn ein Algorithmus existiert, der $L$ entscheidet, sodass die Laufzeit des Algorithmus durch $O(p(\kappa(x), |x|))$ für ein Polynom $p$ beschränkt ist. ■

Mit Satz 3.5 ergibt sich die Inklusionsfolge

$$\text{PPT} \subset \text{FPT} \subset \text{XP}.$$

Diese Klassen bilden die zentralen Klassen zur Untersuchung von parametrisierten Problemen bezüglich ihrer Zeitkomplexität. Eine zu diesen Klassen korrespondierende Hierarchie von Platzkomplexitätsklassen wird an dieser Stelle eingeführt. Sie besteht aus der Klasse XL, einer Entsprechung von XP, FPL, der Klasse der *Fixed-Parameter-Logspace-Problems*, und PLS, der *Pseudo-Logarithmic-Space-Problems*.

■ DEFINITION 3.7. Ein parametrisiertes Problem $(L, \kappa)$ ist in der Klasse XL, wenn für alle $k \geq 1$ gilt, dass $L_k = \{x \mid x \in L \text{ und } \kappa(x) = k\} \in \mathrm{L}$ via eines festen Algorithmus. Die zu XL korrespondierende Klasse der Funktionsprobleme sei FXL. ■

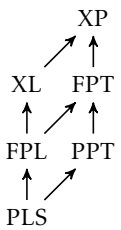
■ DEFINITION 3.8. Ein parametrisiertes Problem $(L, \kappa)$ ist in der Klasse FPL, wenn ein Algorithmus existiert, der $L$ entscheidet und dessen Speicherplatzbedarf für eine Eingabe $x$ durch $O(\log(f(\kappa(x))) + \log|x|)$ für eine berechenbare Funktion $f$ beschränkt ist. Die korrespondierende Klasse der Funktionsprobleme sei FFPL. ■

■ DEFINITION 3.9. Ein parametrisiertes Problem $(L, \kappa)$ ist in der Klasse PLS, wenn ein Algorithmus existiert, der $L$ entscheidet, sodass der Speicherplatzbedarf des Algorithmus durch eine Funktion aus $O(\log \kappa(x) + \log|x|)$ beschränkt ist. Die korrespondierende Klasse der Funktionsprobleme sei FPLS. ■

Aus den Definitionen der Klassen ergibt sich direkt die Hierarchie

$$\mathrm{PLS} \subseteq \mathrm{FPL} \subseteq \mathrm{XL}.$$

Abbildung 3.1 stellt die eingeführten Klassen in ihren Hierarchien und die Zusammenhänge zwischen den Hierarchien dar. Die Inklusionen zwischen den Platz- und Zeitklassen lassen sich direkt über die Größe der Konfigurationsgraphen und die damit verbundene maximale Laufzeit einer platzbeschränkten Maschine ableiten.




■ ABBILDUNG 3.1. Die obige Graphik stellt die bekannten Inklusionen der vorgestellten Komplexitätsklassen dar. Ein Pfeil von einer Klasse $A$ zu einer Klasse $B$ zeigt die Inklusion $A \subseteq B$ an.

Eine für diese Komplexitätsklassen geeignete Reduktion ist die FPLS-Reduktion.

■ DEFINITION 3.10. Es seien $(L_1, \kappa_1)$ und $(L_2, \kappa_2)$ zwei parametrisierte Probleme mit $L_1 \subseteq \mathrm{Struc}(\tau_1)$ und $L_2 \subseteq \mathrm{Struc}(\tau_2)$. Dann ist $f$ eine FPLS-*Reduktion* von $(L_1, \kappa_1)$ auf $(L_2, \kappa_2)$, wenn

1. für alle $x \in \mathrm{Struc}(\tau_1)$ gilt, dass $x \in L_1$ gdw. $f(x) \in L_2$,
2. $(f, \kappa_1) \in$ FPLS und
3. ein Polynom $g$ existiert, sodass $\kappa_2(f(x)) \leq g(\kappa_1(x))$ gilt. ■

■ LEMMA 3.11. *Die Klasse* PLS *ist unter Reduktion abgeschlossen. Die Klasse* FPLS *ist unter Komposition abgeschlossen.* ■

■ BEWEIS. Sei $f$ eine FPLS-Reduktion von $(L_1, \kappa_1)$ auf $(L_2, \kappa_2)$, $\kappa_2(x) \leq g(\kappa_1(x))$ für ein Polynom $g$ und $(L_2, \kappa_2) \in$ PLS. Es ist nun zu zeigen, dass eine Turingmaschine $M$ existiert, die $(L_1, \kappa_1)$ entscheidet und dabei höchstens Speicherplatz $O(\log \kappa_1(x) + \log|x|)$ benötigt.

Diese Turingmaschine kann nicht zunächst $f(x)$ berechnen und dann $f(x) \in L_2$ entscheiden, da $f(x)$ zu lang sein kann, um von $M$ unter Einhaltung der Platzschranke $O(\log \kappa_1(x) + \log|x|)$ gespeichert zu weden. Daher simuliert $M$ eine Turingmaschine $M_{L_2}$ für $(L_2, \kappa_2)$ auf $f(x)$ und berechnet jedesmal, wenn $M_{L_2}$ ein Symbol von $f(x)$ benötigt, dieses neu, indem sie $f(x)$ berechnet und nur das von $M_{L_2}$ geforderte Symbol ausgibt. Hierfür benötigt sie den Speicherplatz für den Index des aktuell zu berechnenden Symbols von $f(x)$. Anschließend simuliert sie $M_{L_2}$ weiter. Auf diese Weise muss $M$ nicht den ganzen String $f(x)$ zwischenspeichern, sondern benötigt nur den Speicherplatz zur Berechnung von $f(x)$, den Speicherplatz für den Index des aktuell angeforderten Symbols von $f(x)$ und den zur Berechnung von $M_{L_2}$ auf $f(x)$:

$$
\begin{aligned}
& O\left(\log \kappa_1(x) + \log|x| + \log 2^{\log \kappa_1(x) + \log|x|} + \log \kappa_2(f(x)) + \log|f(x)|\right) \\
\leq\ & O\left(2\log \kappa_1(x) + 2\log|x| + \log g(\kappa_1(x)) + \log 2^{\log \kappa_1(x) + \log|x|}\right) \\
=\ & O\left(3\log \kappa_1(x) + 3\log|x| + \log g(\kappa_1(x))\right) \\
=\ & O\left(\log\left(g(\kappa_1(x))\right) + \log|x|\right) \\
=\ & O\left(\log \kappa_1(x) + \log|x|\right).
\end{aligned}
$$

■

# KAPITEL 4

## ANWENDUNGEN VON MONADISCHER LOGIK ZWEITER STUFE UND BESCHRÄNKTER BAUMWEITE AUF PLATZKOMPLEXITÄT

Dieses Kapitel enthält Untersuchungen ausgewählter Probleme mittels der in Abschnitt 2.4 gegebenen Sätze von Bodlaender und Courcelle für logarithmischen Platz und dem in Kapitel 3 eingeführten Framework.

Abschnitt 4.1 befasst sich mit *Zahlenproblemen*. Hier werden Probleme über den natürlichen Zahlen $\{0,1,\ldots,n-1\}$ oder den ganzen Zahlen $\{-m,-m+1,\ldots,-1,0,1,\ldots,n-1\}$ für natürliche Zahlen $m$ und $n$ betrachtet. Viele Zahlenprobleme lassen sich durch Graphen mit Baumweite 1 modellieren und dann über MSO-Formel auswerten, wie im Folgenden an Varianten von Problemen wie PARTITION, SUBSET-SUM, BIN-PACKING, KNAPSACK und INTEGER-PROGRAMMING gezeigt wird. Die gewählten Modellierungen und die Anwendung des Satzes von Courcelle führt zu einer natürlichen Parametrisierung dieser Probleme über den Wert der größten in der Eingabe kodierten Zahl, wie sie bereits bei früheren Betrachtungen von schwach-NP-vollständigen erkennbar war (19, 37).

Abschnitt 4.2 befasst sich mit der Untersuchung von Scheduling-Problemen, also der Frage, ob eine gegebenen Menge von Aufgaben unter Beachtung von Start- und Endzeiten bearbeitet werden können. In dieser Arbeit wird eine logische Struktur zur Modellierung von Instanzen des Scheduling-Problems eingeführt und eine parametrisierte Variante des Scheduling-Problems mit den Sätzen von Bodlaender und Courcelle untersucht.

In Abschnitt 4.3 werden *Graphenprobleme* behandelt. Die hier betrachteten Probleme lassen sich über die Baumweite der gegebenen Graph-Instanzen FXL-parametrisieren, was nicht der Definition von parametrisierten Probleme in dieser Arbeit entspricht, aber für die Anwendung der Sätze von Bodlaender und Courcelle sinnvoll ist. Zu den hier betrachteten Problemen gehören CLIQUE, VERTEX-COVER und CHROMATIC-NUMBER.

Abschnitt 4.4 befasst sich mit dem *Erfüllbarkeitsproblem* SATISFIABILITY für aussagenlogische Formeln. Diese Formeln lassen sich als Graphen modellieren, deren Baumweiten von den Variablenvorkommnissen in den Formeln abhängig sind. Diese Abhängigkeiten werden in dieser Arbeit untersucht und mittels dieser Untersuchungen das Erfüllbarkeitsproblem bezüglich der Variablenvorkommnisse parametrisiert.

In Abschnitt 4.5 werden *Mengenprobleme* untersucht. Dabei handelt es sich um Probleme, die sich mit den Eigenschaften von Familien gegebener Mengen beschäf-

tigen. Diese Probleme lassen sich als Graphen modellieren, deren Baumweiten von der Verteilung der Elemente in den Mengen abhängig sind. Diese Abhängigkeiten werden in in dieser Arbeit untersucht und hierüber eine Parametrisierung dieser Probleme angegeben. Beispiele für Mengenprobleme sind SET-COVERING, EXACT-COVER und HITTING-SET.

Die betrachteten Probleme sind eine Auswahl aus Karps 21 NP-vollständigen Problemen (30), schwach-NP-vollständigen Problemen aus dem Buch von Garey und Johnson (19) sowie Probleme aus der aktuellen Literatur.

## 4.1 ZAHLENPROBLEME

Zahlenprobleme sind Probleme, bei denen das betrachtete Universum aus den natürlichen Zahlen $\{0,1,\ldots,n\}$ oder den ganzen Zahlen $\{-m,-m+1,\ldots,-1,0,1,\ldots,n-1\}$ für natürliche Zahlen $m$ und $n$ besteht. Bei den hier betrachteten Zahlenproblemen handelt es sich um schwach-NP-vollständige Probleme, für die gezeigt werden kann, dass ihre parametrisierten Varianten Elemente der Klasse PLS sind.

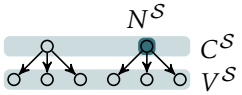
Einer Turingmaschine werden natürliche Zahlen typischerweise in ihrer Binärkodierung zur Berechnung bereitgestellt, was einer natürlichen und platzeffizienten Kodierungsform entspricht. Sollen nun Aussagen über diese Zahlen in MSO-Logik gemacht werden, um die Sätze von Bodlaender und Courcelle anwenden zu können, müssen zwei Aspekte beachtet werden:

1. Die Größe des von den Binärzahlen aufgespannten Universums ist exponentiell in der Länge seiner Kodierung. Dies hat zur Folge, dass aus der erfolgreichen Anwendung des Satzes von Courcelle kein logarithmischer Platzbedarf, sondern nur ein polynomieller Platzbedarf gefolgert werden kann.
2. Die Baumweite der betrachteten Struktur wächst sehr schnell mit der Größe des Universums, beispielsweise durch mathematische Operationen über dem Universum, welche typischerweise durch Relationen ausgedrückt werden. Für eine Anwendung der Sätze von Bodlaender und Courcelle muss jedoch eine beschränkte Baumweite zugesichert werden.

Um diesen Problemen zu begegnen wird in den folgenden Betrachtungen eine Eingabe aus binär kodierten Zahlen in eine logische Struktur umgewandelt, die eine Baumweite von 1 hat und die die Auswertung von gewissen arithmischen Operationen unter der Verwendung von MSO-Formeln ermöglicht. Dies wird jedoch auf Kosten der Größe der logischen Struktur geschehen, denn diese ist exponentiell in der Länge der Binärkodierung der gegebenen natürlichen Zahlen.

Eine natürliche Zahl $a$ wird durch einen Graphen, einem *Stern*, kodiert, der aus einem *Verbindungsknoten* und $a$ *Wertknoten* besteht, wobei die Wertknoten mit dem Verbindungsknoten über eine Kante verbunden ist. Die Knoten eines Sternes werden in Abhängigkeit ihres Typs gefärbt, dargestellt durch das Zusammenfas-

sen von Knoten in Relationen. Die Wertknoten werden in einer Relation $V^{\mathcal{S}}$, die Verbindungsknoten in einer Relation $C^{\mathcal{S}}$ zusammengefasst. Eine negative ganze Zahl entspricht in ihrer Darstellung einer natürlichen Zahl, deren Verbindungsknoten mit einer weiteren Farbe gefärbt ist, dargestellt durch die Relation $N^{\mathcal{S}}$. Abbildung 4.1 zeigt solche Sterne.




- ABBILDUNG 4.1. Die natürliche Zahl 3 wird links durch einen Stern repräsentiert. Die Relation $C^{\mathcal{S}}$ beinhaltet den Verbindungsknoten und die Relation $V^{\mathcal{S}}$ die Wertknoten der Struktur $\mathcal{S}$. Rechts ist die Darstellung der ganzen Zahl −3 dargestellt. Ihr Verbindungsknoten ist zusätzlich gefärbt, was durch die Relation $N^{\mathcal{S}}$ repräsentiert wird.

Die Berechnung einer solchen logischen Struktur sei im Folgenden das parametrisierte Funktionsproblem $p_{\max}$-STAR-STRUCTURE:

- $p_{\max}$-STAR-STRUCTURE.
  - INSTANZ. Eine binär kodierte Folge von ganzen Zahlen.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - ERGEBNIS. Eine logische Struktur aus Sternen, die die gegebenen Zahlen darstellen.
- LEMMA 4.1.

  $p_{\max}$-*STAR-STRUCTURE* $\in$ FPLS. ∎

- BEWEIS. Eine Turingmaschine, die für eine binär kodierte Folge $x$ von ganzen Zahlen eine logische Strukturen aus Sternen ausgibt, benötigt zur Ausgabe der Kodierung eines Sterns einen Zähler mit $O(\log \kappa(x))$ Bits, mit dem die Anzahl der Wertknoten gezählt wird. Weiter enthält eine Eingabe $x$ höchstens polynomiell viele Binärzahlen. Es reicht also ein binärer Zähler mit $O(\log|x|)$ Bits, um die Sterne für die Zahlen auszugeben. Es ergibt sich ein Platzverbrauch von $O(\log|x| + \log \kappa(x))$. ∎

Die verwendete Parametrisierung über den Wert der größten in der Eingabe kodierten Zahl entspricht einer natürlichen Parametrisierung, wie sie bereits bemm Konzept der schwach-NP-vollständigen Probleme zu erkennen ist (19). All in diesem Abschnitt betrachteten Zahlenprobleme lassen sich so parametrisieren und werden mit dem Präfix $p_{\max}$- gekennzeichnet.

Nicht nur einzelne Zahlen können durch Sterne repräsentiert werden, sondern auch Tupel von Zahlen und Mengen von Zahlen. Ein Tupel mit $k$ Komponenten $(c_1, c_2, \ldots, c_k)$ wird durch $k$ Sterne kodiert, deren Verbindungsknoten mit zusätzlichen Farben $C_1^{\mathcal{S}}, \ldots, C_k^{\mathcal{S}}$ gefärbt sind. Diese Verbindungsknoten werden durch

einen zusätzlichen Verbindungsknoten, den *Tupelknoten* über jeweils eine Kante verbunden. Tupelknoten werden in der Relation $T^{\mathcal{S}}$ zusammengefasst. Eine Menge wird wie ein Tupel kodiert, bei dem die Verbindungsknoten gleich und der Tupelknoten als *Mengenknoten* durch $S^{\mathcal{S}}$ gefärbt ist. Abbildung 4.2 zeigt jeweils ein Beispiel eines Tupels und einer Menge.


- ABBILDUNG 4.2. Das Tupel $(2,3)$ wird durch den Stern links repräsentiert. Die Relationen $C_1^{\mathcal{S}}$ und $C_2^{\mathcal{S}}$ beinhalten Verbindungsknoten der Sterne für die Zahlen 2 und 3, die Relation $V^{\mathcal{S}}$ beinhaltet die Wertknoten. Die Menge $\{2,3\}$ wird durch den Stern rechts dargestellt. Dieser Stern entspricht dem Stern links bis auf eine einheitliche Färbung der Verbindungsknoten und eine andere Färbung des Tupelknotens.

Mit diesen Strukturen können viele Zahlenprobleme parametrisiert und ausgewertet werden, indem der Parameter dem Wert der größten in der Eingabe kodierten Zahl entspricht und unter Verwendung von Lemma 4.1 eine logische Struktur aus Sternen konstruiert wird. Für diese Struktur wird das Histogramm einer zum Problem korrespondierenden MSO-Formel berechnet, über dessen Auswertung das Problem entschieden werden kann. Die Auswertung der MSO-Formel ist hierbei auf logarithmischem Platz in der Größe des Univerums der logischen Struktur aus Sternen möglich, da diese immer die Baumweite 1 hat und somit ein Problem der Klasse FPLS mit der Parametrisierung $\kappa(x) = 1$ ist. Insgesamt kann auf diese Weise für parametrisierte Zahlenprobleme gezeigt werden, dass sie in der Klasse PLS sind, da es sich hierbei um eine FPLS-Reduktion auf ein Problem der Klasse PLS handelt. Dies wird im Folgenden an der parametrisierten Zählvariante des Problems PARTITION genauer betrachtet.

- $p_{\max}$-COUNT-PARTITIONS.
  - INSTANZ. Eine Menge von natürlichen Zahlen.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - ERGEBNIS. Die Anzahl der Partionierungen der Zahlen in zwei Mengen, sodass die jeweiligen Elementsummen gleich sind.
- SATZ 4.2.

  $p_{\max}$-*COUNT-PARTITIONS* $\in$ FPLS. ∎

- BEWEIS. Zunächst wird aus der gegebenen Liste von Zahlen eine Stern-Struktur konstruiert. Dieses Funktionsproblem ist nach Lemma 4.1 in der Klasse FPLS. Anschließend wird eine MSO-Formel $\phi_{\text{PARTITION}}$ mit folgenden Eigenschaften ausgewertet: Die Formel $\phi_{\text{PARTITION}}$ hat zwei freie einstellige Variablen $S_1$ und $S_2$ zweiter

Stufe und drückt aus, dass eine Partitionierung der Verbindungsknoten $C^{\mathcal{S}}$ und damit der Wertknoten $V^{\mathcal{S}}$ in zwei Mengen $V_1$ und $V_2$ existiert, deren Wertknoten Elemente von $S_1$ und $S_2$ sind:

$$\begin{aligned}
\phi_{\text{PARTITION}}(S_1, S_2) &= \exists C_1 C_2 . \varphi_{C\text{-PART}}(C_1, C_2) \wedge \varphi_{V\text{-PART}}(S_1, S_2), \\
\varphi_{C\text{-PART}}(C_1, C_2) &= \forall x . C(x) \to [C_1(x) \vee C_2(x)] \wedge \neg[C_1(x) \wedge C_2(x)], \\
\varphi_{V\text{-PART}}(S_1, S_2) &= \forall xy . E(x,y) \to [C_1(x) \leftrightarrow S_1(y)] \wedge [C_2(x) \leftrightarrow S_2(y)].
\end{aligned}$$

Die Formel $\phi_{\text{PARTITION}}$ mit den zwei Variablen $S_1$ und $S_2$ ist genau dann wahr, wenn eine Partitionierung der Verbindungsknoten $C^{\mathcal{S}}$ in zwei Mengen $C_1$ und $C_2$ existiert und die Mengen $S_1$ und $S_2$ die Menge der Wertknoten $V^{\mathcal{S}}$ so partitioniert, dass ein Wertknoten genau dann in der Menge $S_i$ ist, wenn sein zugehöriger Verbindungsknoten in der Menge $C_i$ ist. Die Formel $\varphi_{C\text{-PART}}$ stellt die korrekte Partitionierung der Verbindungsknoten und $\varphi_{V\text{-PART}}$ die der Wertknoten sicher.

Existieren zwei Mengen $S_1$ und $S_2$ von gleicher Kardinalität, sodass die Formel $\phi_{\text{PARTITION}}(S_1, S_2)$ gilt, dann lässt sich die gegebene Menge von Zahlen wie gefordert partitionieren. Die Anzahl solcher Partitionierungen lässt sich aus dem Lösungshistogramm $\text{histogram}[\mathcal{S}, \phi_{\text{PARTITION}}]$ berechnen: Der Eintrag an der Stelle $(|V^{\mathcal{S}}|/2, |V^{\mathcal{S}}|/2)$ im Histogramm entspricht der doppelten Anzahl der möglichen Partitionierungen, wobei $|V^{\mathcal{S}}|$ die Anzahl der Wertknoten der Struktur $\mathcal{S}$ angibt. Es handelt sich um die doppelte Anzahl, da für alle Partitionierungen eine symmetrische Partitionierung existiert, die im Histogramm gezählt wird.

Die hierbei konstruierten Graph-Strukturen haben jeweils die Baumweite 1. Damit ist die Auswertung der MSO-Formel nach Satz 2.28 in logarithmischem Platz in der Größe des Universums der Stern-Struktur möglich. Nach Lemma 4.1 ist diese Größe in $O(|x| \cdot \kappa(x))$. Die Konstruktion des Lösungshistogramm ist damit FPLS-berechenbar. Die anschließende Suche nach einem positiven Eintrag im Histogramm benötigt einen Zähler, dessen Speicherplatzbedarf logarithmisch in der Größe des Univerums der Stern-Struktur ist. Nach Lemma 3.11 gilt dann $p_{\max}$-COUNT-PARTITIONS $\in$ FPLS. ∎

Für das klassische Entscheidungsproblem PARTITION lässt sich mit diesem Satz leicht zeigen, dass es in PLS liegt.

■ KOROLLAR 4.3.

$p_{\max}$-*PARTITION* $\in$ PLS. ∎

Das in diesem Beweis gezeigte Vorgehen lässt sich unter kleinen Änderungen auch bei den im Folgenden betrachteten Problemen einsetzen.

Das Problem SUBSET-SUM gehört zu den bekanntesten Problemen der Kryptographie und der Komplexitätstheorie. In der Kryptographie wird die NP-Vollständigkeit des Problems zur Erzeugung von *Pseudo-Zufallszahlengeneratoren* und

*One-Way-Hash-Funktionen* genutzt (35, 25). Auch hier wird die parametrisierte Zählvariante betrachtet.

- $p_{\max}$-COUNT-SUBSET-SUMS.
  - INSTANZ. Eine Menge von natürlichen Zahlen und eine Zielsumme $s$.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - ERGEBNIS. Die Anzahl der Teilmengen der gegebenen Zahlen, sodass ihre Summe der Zielsumme $s$ entspricht.

- SATZ 4.4.

  $p_{\max}$-*COUNT-SUBSET-SUMS* $\in$ FPLS. ■

- BEWEIS. Der Beweis entspricht dem von Satz 4.2 mit folgenden Änderungen:
  1. Für jede gegebene Zahl bis auf die Zielsumme $s$ wird ein Stern konstruiert, der diese Zahl repräsentiert.
  2. Eine MSO-Formel mit einer einstelligen Variablen zweiter Stufe drückt aus, dass eine Teilmenge der Sterne existiert, sodass die freie Variable genau die Wertknoten dieser Sterne beinhaltet.
  3. Der Wert des Histogramms am Index $s$ wird ausgegeben, denn dieser entspricht der Anzahl der Teilmengen, deren Summe genau $s$ ergibt. ■

Für das Entscheidungsproblem SUBSET-SUM ergibt sich damit sofort das folgende Korollar.

- KOROLLAR 4.5.

  $p_{\max}$-*SUBSET-SUM* $\in$ PLS. ■

Eine schwach-NP-vollständige Variante von SUBSET-SUM findet sich bei Garey und Johnson unter dem Namen $k$-LARGEST-SUBSET (19), bei der für ein gegebenes $k$ entschieden werden soll, ob $k$ oder mehr verschiedene Teilmengen $A$ existieren, sodass $\sum_{x \in A} x \leq s$ gilt. Auch für dieses Problem ergibt sich mit dem obigen Satz das folgende Korollar.

- KOROLLAR 4.6.

  $p_{\max}$-$k$-*LARGEST-SUBSET* $\in$ PLS. ■

Durch das Lösungshistogramm kann nicht nur die Anzahl der Teilmengen ermittelt werden, deren jeweilige Summe einem Zielwert entsprechen, sondern weiter auch deren Kardinalität.

- $p_{\max}$-COUNT-$m$-SUBSET-SUMS. Für eine gegebene Menge von Zahlen soll berech-

net werden, wie viele Teilmengen der gegebenen Zahlenmenge mit Kardinalität $m$ existieren, deren Summe einer gegebene Maximalsumme $s$ entspricht.

- INSTANZ. Eine Menge von natürlichen Zahlen, eine natürliche Zahl $m$ und eine natürliche Zahl $s$.
- PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
- ERGEBNIS. Die Anzahl der Teilmengen $A$ der gegebenen Zahlenmenge mit Kardinalität $m$, sodass $\sum_{x \in A} x = s$ gilt.

- SATZ 4.7.

  $p_{\max}$-*COUNT-m-SUBSET-SUMS* $\in$ FPLS. ∎

- BEWEIS. Der Beweis entspricht dem von Satz 4.4 mit folgenden Änderungen:
  1. Für jede gegebene Zahl bis auf $m$ und $s$ wird ein Stern konstruiert, der die Zahl repräsentiert.
  2. Eine MSO-Formel mit zwei freien einstelligen Variablen drückt aus, dass die erste Variable eine Teilmenge der Verbindungsknoten der Sterne und die zweite Variable genau die zu diesen Sternen korrespondierenden Wertknoten beinhaltet.
  3. Ein Eintrag am Index $(m, s)$ im Histogramm gibt an, wie viele unterschiedliche Mengen von $m$ Sternen mit $s$ Wertknoten, also wie viele Mengen von $m$ Zahlen mit Summe $s$, existieren. Dieser Eintrag wird ausgegeben. ∎

Eine Verallgemeinerung von SUBSET-SUM ist das Rucksackproblem KNAPSACK, für das unterschiedlichste Varianten definiert und untersucht worden sind (33). Bei der im Folgenden betrachteten Variante handelt es sich um die parametrisierte Zählversion der bekanntesten Variante, $p_{\max}$-COUNT-0-1-KNAPSACKS.

- $p_{\max}$-COUNT-0-1-KNAPSACKS. Für eine gegebene Menge von Objekten mit jeweils einem Gewicht und einem Wert sowie einen gegebenen Rucksack, der ein maximales Gewicht tragen kann, soll für einen Zielwert berechnet werden, wie viele Auswahlmöglichkeiten für Elemente existieren, sodass jede Auswahl mit dem Rucksack getragen werden kann und zugleich mit ihrem Gesamtwert mindestens dem Zielwert entspricht.
  - INSTANZ. Eine Liste von Zahlentupeln $(v_i, w_i)$ für die Objekte, deren erste Komponente den Wert und deren zweite Komponente das Gewicht des Objekts angeben, ein Zielwert $v$ und ein maximales Tragegewicht des Rucksacks $w$.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - ERGEBNIS. Die Anzahl der Teilmengen $B$ von Zahlentupeln, sodass jeweils $\sum_{(v_i, w_i) \in B} v_i \geq v$ und $\sum_{(v_i, w_i) \in B} w_i \leq w$ gilt.

- SATZ 4.8.

  $p_{\max}$-*COUNT*-0-1-*KNAPSACKS* $\in$ FPLS. ∎

■ BEWEIS. Der Beweis entspricht dem von Satz 4.2 mit folgenden Änderungen:

1. Für jedes gegebene Objekt wird ein Tupel-Stern konstruiert, der den Wert und das Gewicht des Objekts repräsentiert.
2. Eine MSO-Formel mit zwei einstelligen Variablen $V_{\text{Bag}}$ und $W_{\text{Bag}}$ drückt aus, dass eine Teilmenge der Tupel-Sterne existiert, sodass $V$ genau deren Wertknoten und $W$ genau deren Gewichtsknoten enthält.
3. Aus dem Histogramm wird die Summe der Einträge für Indizes $(i_1, i_2)$ mit $i_1 \geq v$ und $i_2 \leq w$ berechnet. Diese Summe entspricht genau der Anzahl der Auswahlmöglichkeiten von Elementen zur Befüllung des Rucksacks, die jeweils die gegebenen Bedingungen erfüllen. ■

Für das klassische Problem 0-1-KNAPSACK ergibt sich damit sofort das folgende Korollar.

■ KOROLLAR 4.9.

$p_{\max}$-0-1-*KNAPSACK* $\in$ PLS. ■

Analog zu Satz 4.7 lässt sich auch für das Rucksackproblem eine Variante definieren, bei der berechnet wird, wie viele zulässige Packungen des Rucksacks mit einer gegebenen Anzahl von Elementen existieren. Weiter lässt sich aus den Beweisen von Satz 4.7 und Satz 4.8 das folgende Korollar ableiten.

■ KOROLLAR 4.10.

$p_{\max}$-*COUNT*-$m$-0-1-*KNAPSACKS* $\in$ FPLS. ■

Eine allgemeinere Form des Problems KNAPSACK ist das im Folgenden betrachtete Problem INTEGER-PROGRAMMING.

Die Lineare Programmierung gehört zweifelsohne zu den bekanntesten Optimierungsproblemen. Der Anwendungsbereich erstreckt sich von der reinen Optimierung über ökonomische Planung bis zur Lösung von kombinatorischen Problemen. Die Untersuchungen zur linearen Optimierung und Veröffentlichung des Simplex-Algorithmus durch Dantzig führten zum Durchbruch der linearen Programmierung. Bis heute sind zahlreiche Fortschritte im Bereich der linearen Optimierung entwickelt worden (20).

Lineare Optimierung beschäftigt sich mit der Optimierung von linearen Zielfunktionen unter Beachtung linearer Gleichungs- und Ungleichungssysteme, die die Optimierung einschränken. In der oft als *Standardform* bezeichneten Variante ist eine $m \times n$-Matrix $A$, ein $m$-Vektor $b$ und ein $n$-Vektor $c$ gegeben. Gesucht ist eine Lösung $x$ für das Gleichungssystem $Ax = b$ über den natürlichen Zahlen, sodass $c^T x$ minimal wird. Alternative Formen wie die *kanonische Form*, bei der ein Ungleichungssystem $Ax \geq b$ zu lösen ist, die *Standardform mit Ungleichungen* mit einem Ungleichungssystem $Ax \leq b$ oder die *generelle Form*, in der sowohl Ungleichungen

als auch Gleichungen gegeben sind, lassen sich durch einfache Transformationen in die Standardform überführen. Hierfür wird für jede Ungleichung eine neue Variable, eine *Schlupf-Variable*, und eine neue Gleichung eingefügt. Genaueres hierzu findet sich im Buch von Gill (20). Aufgrund dieser einfachen Transformationen wird im folgenden nur die Standardform weiter untersucht. Die Transformationen der anderen Formen in die Standardform beeinflussen dabei die Ergebnisse dieser Arbeit nicht. Weiter wird statt des Optimierungsproblems zunächst ein parametrisiertes Entscheidungsproblem $p$-INTEGER-PROGRAMMING in seiner Formulierung nach Karp (30) und Papadimitriou (36) untersucht.

- $p_{\max}$-INTEGER-PROGRAMMING.
  - INSTANZ. Eine $m \times n$-Matrix $A$ über den ganzen Zahlen und ein $m$-Vektor $b$ über den ganzen Zahlen.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - FRAGE. Existiert ein $n$-Vektor $x$ über den natürlichen Zahlen, sodass $Ax = b$ gilt?

Dieses Problem ist stark-NP-vollständig und für eine fest gewählte Anzahl von Zeilen der Matrix schwach-NP-vollständig (19, 30, 7, 36, 37). Die Variante mit fest gewählter Zeilenzahl sei $p_{\max}$-FIXED-$m$-INTEGER-PROGRAMMING.

Wie bisher wird eine gegebene Instanz von $p_{\max}$-FIXED-$m$-INTEGER-PROGRAMMING zunächst in eine logische Struktur transformiert und für diese dann eine MSO-Formel und ihr Histogramm ausgewertet. Bisher wurden die gegebenen Zahlen in Stern-Strukturen transformiert. Das ist nun nicht mehr hinreichend, da die gegebenen Zahlen nicht unabhängig voneinander sind, sondern in der Matrix $A$ und dem Vektor $b$ einer Struktur unterliegen. Dies wird in der im Folgenden vorgestellten Struktur zur Lösung des Problems beachtet. Weiter wird diese Struktur so ausgestaltet, dass das Ergebnis der Multiplikation der Matrix $A$ mit einem Lösungsvektor $x$ aus dem Histogramm einer MSO-Formel $\phi_{\text{IP}}$ abgelesen werden kann. Diese *Integer-Programming-Struktur* wird im Folgenden anhand des Beispiels in Abbildung 4.3 beschrieben.

Für jeden Eintrag $a_{i,j}$ der Matrix wird zunächst ein Stern konstruiert. Die Verbindungsknoten der mit den Spalten korrespondierenden Sternen werden durch einen ungerichteten Pfad unter Beachtung der Reihenfolge der Zeilen miteinander verbunden. Für jede Spalte der Matrix werden $r = n(ma)^{2m+1}$ Kopien dieser zu *Stern-Pfaden* verbundenen Sterne erstellt, wobei $a$ dem Wert des größten Eintrags von $A$ und $b$ entspricht. Die Zahl $r$ wird im Anschluss an die Beschreibung der Struktur und der Lösbarkeit von $p_{\max}$-FIXED-$m$INTEGER-PROGRAMMING diskutiert. Die zu einer Spalte gehörenden Stern-Pfade werden durch einen gerichteten Pfad über die zur ersten Zeile der Matrix korrespondierenden Knoten verbunden. Zusätzlich existiert für jeden dieser gerichteten *Verbindungspfade* ein Knoten, der *Variablenknoten*, der durch eine gerichtete Kante mit dem ersten Knoten des Ver-

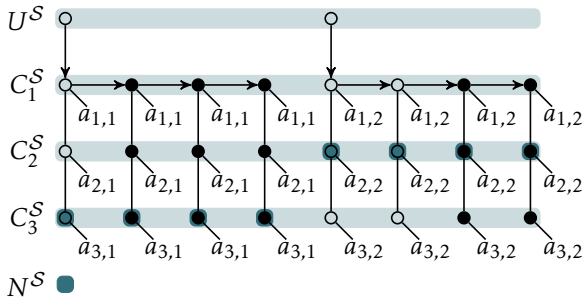



ABBILDUNG 4.3. Die obige Abbildung zeigt die für eine $3 \times 2$-Matrix $A$ konstruierte Integer-Programming-Struktur mit $r = 4$. Aus Gründen der Übersichtlichkeit werden die Wertknoten in dieser Abbildung durch Einträge $a_{i,j}$ abgekürzt. Die durch die freie Variable $X$ gewählten Verbindungsknoten der Zeile $C_1^{\mathcal{S}}$ induzieren eine Knotenfärbung, dargestellt durch Knoten ●, die der Wahl von $x^T = (3, 2)$ entspricht. Die Variable $X$ enthält für die erste Spalte der Matrix den zweiten Knoten und für die zweite Spalte den dritten Knoten des jeweiligen Verbindungspfades. Die durch diese Färbung induzierten Kardinalitäten der freien Variablen $C_i^+$ und $C_i^-$ für $i \in \{1, 2, 3\}$ der Formel $\phi_{\text{IP}}$ sind $|C_1^+| = 3 \cdot a_{1,1} + 2 \cdot a_{1,2}$, $|C_1^-| = 0$, $|C_2^+| = 3 \cdot a_{2,1}$, $|C_2^-| = 2 \cdot a_{2,2}$, $|C_3^+| = 2 \cdot a_{3,2}$ und $|C_3^-| = 3 \cdot a_{3,1}$.

bindungspfades verbunden ist. Die Variablenknoten werden in der einstelligen Relation $U^{\mathcal{S}}$ und die mit der $i$-ten Zeile der Matrix korrespondierenden Verbindungsknoten der Sterne in der einstelligen Relation $C_i^{\mathcal{S}}$ zusammengefasst, sodass die Knoten der Relation $C_1^{\mathcal{S}}$ den Knoten der Verbindungspfade entsprechen. Diese Struktur hat die Baumweite 1, da sie aus Bäumen besteht, und lässt sich unter Beachtung von Lemma 4.1 ebenfalls in FPLS berechnen.

SATZ 4.11.

$p_{\max}$-*FIXED-M-INTEGER-PROGRAMMING* $\in$ PLS. ■

BEWEIS. Zunächst wird aus der Eingabe die Integer-Programming-Struktur konstruiert. Anschließend wird das Histogramm einer MSO-Formel $\phi_{\text{IP}}$ mit folgenden Eigenschaften ausgewertet.

1. Die freien Variablen der Formel sind $X$ und $C_i^+$, $C_i^-$ für $i \in \{1, \ldots, m\}$. Die Variable $X$ ist eine Teilmenge der Knoten der Verbindungspfade, sodass für jeden Variablenknoten genau ein Knoten aus dem zugehörigen Verbindungspfad in $X$ ist. Dies kann über den transitiven Abschluss aus Lemma 2.11 und die Variablenknoten in $U^{\mathcal{S}}$ ausgedrückt werden. Diese Knoten induzieren eine eindeutige Färbung des Graphen: Jeder Knoten aus $X$ und jeder von einem Knoten aus $X$ erreichbare Verbindungsknoten ist gefärbt. Auch dies lässt sich über den transitiven Abschluss ausdrücken.
2. Die freien Variablen $C_i^+$ enthalten dann die Wertknoten der gefärbten Sterne,

deren Verbindungsknoten in $C_i^S$ sind und die positive Zahlen repräsentieren. Die freien Variablen $C_i^-$ enthalten jene, die negative Zahlen repräsentieren.

Für eine gültige Belegung der freien Variablen der Formel $\phi_{\text{IP}}$ gilt dann, dass die Knoten aus $X$ einen Vektor $x$ definieren, der mit der Matrix $A$ multipliziert wird und dessen Ergebnis aus dem Histogramm ausgelesen werden kann. Wird durch einen Knoten die Färbung von $i$ Stern-Pfaden für Stern-Pfade der Spalte $j$ induziert, so entspricht dies der Multiplikation der $j$-ten Spalte der Matrix $A$ mit der Zahl $i$, da entsprechend viele Wertknoten der Sterne für diese Spalte in den Variablen $C_i^+$ und $C_i^-$ gesammelt werden. Die Anzahl der Wertknoten in $C_i^+$ minus der Anzahl der Wertknoten in $C_i^-$ entspricht dem Skalarprodukt eines durch die Knoten in $X$ beschriebenen Vektors $x$ mit der $i$-ten Zeile der Matrix.

Aus dem Histogramm kann dann abgelesen werden, ob ein Lösungsvektor $x$ für das Gleichungssystem $Ax = b$ existiert: Die Summe aller Einträge an den Indizes $(n, c_1^+, c_1^-, c_2^+, c_2^-, \ldots, c_m^+, c_m^-)$, für die $c_i^+ - c_i^- = b_i$ gilt, entspricht der Anzahl der möglichen Lösungen $x$, sodass für die Komponenten $x_i$ des Lösungsvektors $x_i \in \{0, 1, \ldots, n(ma)^{2m+1}\}$ gilt. Dass die Beschränkung auf Lösungsvektoren $x \in \{0, 1, \ldots, n(ma)^{2m+1}\}$ ausreicht, zeigte Papadimitriou:

■ SATZ 4.12: (36). *Es sei $A$ eine $m \times n$-Matrix und $b$ ein $m$-Vektor mit Einträgen über den Zahlen $\{0, \pm 1, \ldots, \pm a\}$. Hat $Ax = b$ eine Lösung für einen $m$-Vektor $x$ über den natürlichen Zahlen, so existiert auch eine Lösung über den Zahlen $\{0, 1, \ldots, n(ma)^{2m+1}\}$.* ■

Es reicht also aus, jeden Stern-Pfad $n(ma)^{2m+1}$-mal zu erzeugen, da dann, wenn eine Lösung für das Gleichungssystem existiert, eine korrespondierende Färbung des Graphen durch die Knoten in $X$ gefunden werden kann, sodass die Auswertung des Histogramms eine existierende Lösung anzeigt. ■

In den bisherigen Betrachtungen wurde die Optimierung der Zielfunktion nicht beachtet, sondern nur der Nachweis der Existenz einer Lösung unter den gegebenen Einschränkungen untersucht. Papadimitriou zeigte, dass die Kosten einer minimalen Lösung beschränkt sind, falls das Gleichungssystem beschränkt ist.

■ LEMMA 4.13: (36). *Ist das ganzzahlige lineare Optimierungsproblem $Ax = b$ lösbar und sind die optimalen Kosten $z = c^T x$ für dieses System beschränkt, so gilt*

$$|z| \leq \left( \sum_{j=1}^{n} |c_j| \right) \cdot n^2 (ma^2)^{2m+3},$$

*wobei $c_j$ die $j$-te Komponente von $c$ bezeichnet.* ■

Mit diesem Ergebnis kann die folgende Minimierungsvariante gelöst werden, bei der für eine durch einen $n$-Vektor $c$ gegebene Zielfunktion die minimalen Kosten einer Lösung $x$ für $Ax = b$ berechnet werden, falls das System beschränkt ist.

■ SATZ 4.14.

$p_{\max}$-*MINIMIZE-FIXED-m-INTEGER-PROGRAMMING* $\in$ FPLS. ■

- BEWEIS. Der Beweis ist analog zu dem von Satz 4.11 mit folgenden Änderungen:
  1. In die konstruierte Stern-Struktur wird der Lösungsvektor $c$ einkodiert, indem $c$ als eine weitere Zeile der Matrix $A$ betrachtet wird, deren Verbindungsknoten gesondert gefärbt sind.
  2. Die auswertende MSO-Formel wird um zwei freie einstellige Variablen $C^+$ und $C^-$ zweiter Stufe erweitert, in denen analog zu den Variablen $C_i^+$ und $C_i^-$ die Wertknoten der positiven und der negativen, durch einen Lösungsvektor gefärbten Sterne gesammelt werden.
  3. Im Histogramm wird mittels der neuen Variablen geprüft, ob eine Lösung für $Ax = b$ existiert, sodass $|C^+| - |C^-|$ minimal wird. Diese Suche ist nach Lemma 4.13 polynomiell in der Anzahl der Variablen und dem Wert der größten in der Eingabe kodierten Zahl. Somit werden zum Durchsuchen des Histogramms Zähler mit Platzbedarf $O(\log|x| + \log\kappa(x))$ benötigt. Es folgt die Behauptung. ∎

Ein weiteres klassisches Optimierungsproblem ist BIN-PACKING: Eine Menge von Objekten unterschiedlicher Größe sollen in eine möglichst kleine Anzahl von Behältern fester Größe verpackt werden. Dieses stark-NP-vollständige Problem (30, 19) ist insbesondere im Bereich der Approximationsalgorithmen in verschiedene Varianten untersucht worden (12, 28).

Bei der zunächst betrachteten Variante handelt es sich um die parametrisierter Zählversion des schwach-NP-vollständigen Problems $k$-BIN-PACKING.

- $p_{\max}$-COUNT-$k$-BIN-PACKINGS.
  - INSTANZ. Eine Menge von natürlichen Zahlen und eine natürliche Zahl $b$.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - ERGEBNIS. Die Anzahl der Partitionierungen der Zahlen in $k$ Mengen, sodass für jede Menge $S_i$ gilt, dass $\sum_{x\in S_i} x \leq b$.
- SATZ 4.15.

  $p_{\max}$-*COUNT-$k$-BIN-PACKINGS* $\in$ FPLS. ∎

- BEWEIS. Der Beweis entspricht dem von Satz 4.4 mit folgenden Änderungen:
  1. Für jede der gegebenen Zahlen bis auf die Behältergröße $b$ wird ein Stern konstruiert, der die Zahl repräsentiert.
  2. Eine MSO-Formel mit $k$ freien einstelligen Variablen zweiter Stufe $B_1,\ldots,B_k$ drückt aus, dass eine Partitionierung der Sternknoten in $k$ Mengen existiert, sodass die freie Variable $B_i$ genau die Wertknoten der $i$-ten Sternmenge beinhaltet.
  3. Die Summe der Histogrammeinträge an Indizes $(i_1,\ldots,i_k)$ mit $i_j \leq b$ für $j \in \{1,\ldots,k\}$ gibt an, wie viele zulässige Packungen der gegegeben Elemente in $k$ Mengen existieren, wobei hierbei jede mögliche Permutation von Packungen

in die Behälter mitgezählt wird. Die Summe der Einträge durch $k!$ ergibt also die gesuchte Anzahl von Packungen. ■

Für das Entscheidungsproblem $k$-BIN-PACKING ergibt sich damit sofort das folgende Korollar.

- KOROLLAR 4.16.

  $p_{\max}$-*$k$-BIN-PACKING* $\in$ PLS. ■

Über das Lösungshistogramm lässt sich auch die folgende Minimierungs-Variante berechnen.

- $p_{\max}$-MINIMIZE-$k$-BIN-PACKING.
  - INSTANZ. Eine Menge von natürlichen Zahlen und eine natürliche Zahl $b$.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - ERGEBNIS. Die kleinste Anzahl $j \in \{1, 2, \ldots, k\}$ von Partitionen für die gegebene Menge von Zahlen, sodass für jede Partition $S_i$ gilt, dass $\sum_{x \in S_i} x \leq b$.
- SATZ 4.17.

  $p_{\max}$-*MINIMIZE-$k$-BIN-PACKING* $\in$ FPLS. ■

- BEWEIS. Dieser Beweis entspricht dem von Satz 4.15, wobei die kleinste Zahl $l$ ausgegeben wird, für die ein echt positiver Eintrag im Histogramm an einem Index $(i_1, \ldots, i_k)$ existiert, von dem genau $l$ Komponenten nicht Null sind. Diese Zahl entspricht genau der kleinsten Zahl von benötigten Partitionen. ■

Anstatt ausschließlich einzelne Zahlen zu betrachten, können durch die Stern-Darstellung auch Probleme formuliert werden, deren Instanzen Tupel von Zahlen sind. Ein Beispiel für ein Problem, welches mittels der Tupeldarstellung berechnet werden kann, ist das parametrisierte Zählproblem $p_{\max}$-COUNT-SIZE-$m$-TUPLES.

- $p_{\max}$-COUNT-SIZE-$m$-TUPLES.
  - INSTANZ. Eine Familie von $m$ Mengen $X_1, \ldots, X_m$ von natürlichen Zahlen und eine natürliche Zahl $s$.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - ERGEBNIS. Die Anzahl von Tupeln $A \subseteq X_1 \times \cdots \times X_m$, sodass für jedes Tupel $(x_1, \ldots, x_m) \in A$ gilt, $\sum_{i=1}^{m} x_i = s$.
- SATZ 4.18.

  $p_{\max}$-*COUNT-SIZE-$m$-TUPLES* $\in$ FPLS. ■

- BEWEIS. Der Beweis entspricht dem von Satz 4.2 mit folgenden Änderungen:
  1. Für jedes Tupel aus $X_1 \times \cdots \times X_m$ wird ein Tupel-Stern konstruiert, der das Tupel repräsentiert.

2. Eine MSO-Formel mit einer freien einstelligen Variablen zweiter Stufe drückt aus, dass die freie Variable exakt die Wertknoten von genau einem Tupel-Stern beinhaltet.
3. Der Eintrag im Histogramm im Index $s$ gibt an, wie viele Tupel existieren, deren Komponentensumme jeweils $s$ ergibt. ■

Eine schwach-NP-vollständige Variante dieses Problems findet sich bei Garey und Johnson unter dem Namen $k$-LARGEST-$m$-TUPLE (19), bei der für ein gegebenes $k$ entschieden werden soll, ob $k$ oder mehr verschiedene Tupel über der Menge $X_1 \times \cdots \times X_m$ existieren, sodass deren jeweilige Komponentensumme mindestens einer Zielsumme $s$ entspricht. Für dieses Problem ergibt sich mit dem obigen Satz das folgende Korollar.

■ KOROLLAR 4.19.

$p_{\max}$-$k$-*LARGEST*-$m$-*TUPLE* $\in$ PLS. ■

Ein weiteres Anwendungsbeispiel, welches die Vielfalt der Anwendungsmöglichkeiten der Sätze von Bodlaender und Courcelle aufzeigt, ist das Problem $p_{\max}$-COUNT-RANDOMIZATION-TEST-FOR-MATCHED-PAIRS.

■ $p_{\max}$-COUNT-RANDOMIZATION-TEST-FOR-MATCHED-PAIRS.

- INSTANZ. Eine Menge $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ von Paaren natürlicher Zahlen.
- PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
- ERGEBNIS. Die Anzahl von Teilmengen $S$ dieser Tupel, sodass

$$\sum_{i \in S} |x_i - y_i| \leq \sum_{\substack{x_i > y_i, \\ i \in S}} (x_i - y_i) \text{ gilt.}$$

Um dieses Problem zu lösen wird der Aufbau der bisher betrachteten Stern-Strukturen modifiziert. Anstatt wie bisher einen Verbindungsknoten mit Wertknoten zu verbinden, die genau mit einer Farbe gefärbt sind, können Wertknoten nun mehrfach gefärbt sein. Das Tupel $(x, y)$ wird durch einen Stern mit $\max\{x, y\}$ Knoten dargestellt, wobei $x$ davon mit einer Farbe $V_1^{\mathcal{S}}$ und $y$ mit einer Farbe $V_2^{\mathcal{S}}$ gefärbt sind. Auch diese Strukturen haben die Baumweite 1. Abbildung 4.4 zeigt zwei solche Sterne für zwei Tupel.

Diese modifizierten Stern-Strukturen können ebenfalls durch FPLS-Turingmaschinen berechnet werden.

■ SATZ 4.20.

$p_{\max}$-*COUNT-RANDOMIZATION-TEST-FOR-MATCHED-PAIRS* $\in$ FPLS. ■

■ BEWEIS. Für die gegebenen Tupel werden mehrfach gefärbte Tupel-Sterne konstruiert. Die durch die Relation $V_1^{\mathcal{S}}$ gefärbten Wertknoten seien die der ersten Komponente, die durch die Relation $V_2^{\mathcal{S}}$ gefärbten die der zweiten Komponente.

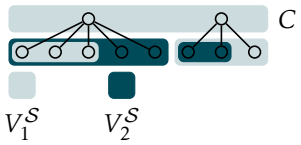



ABBILDUNG 4.4. Die obige Grafik stellt die für die Tupel $(3,5)$ und $(3,2)$ konstruierten Sterne dar. Die Wertknoten der ersten Komponente werden durch die Relation $V_1^S$ und die der zweiten Komponente durch die Relation $V_2^S$ gefärbt.

Durch eine MSO-Formel mit drei freien einstelligen Variablen zweiter Stufe $S$, $A$ und $B$ wird ausgedrückt, dass die Variable $S$ eine Teilmenge der Verbindungsknoten beinhaltet und somit eine Auswahl von Sternen darstellt. Die Variable $A$ enthält von jedem ausgewählten Stern die Wertknoten, die entweder in der Menge $V_1^S$ oder der Menge $V_2^S$ sind, und $B$ enthält alle Wertknoten, die in $V_1^S$, aber nicht in $V_2^S$ sind. Für jeden Stern der ein Tupel $(x,y)$ darstellt, beinhaltet $A$ somit $|x-y|$ und $B$ genau $\sum_{x_i>y_i}(x_i-y_i)$ Wertknoten. Die zugehörige Formel ist

$$\begin{aligned}\phi_{\text{RAND-TEST}}(S,A,B) = \phi_{\subseteq}(S,C) \wedge \\ [\forall u.A(u) \leftrightarrow \exists v.S(v) \wedge E(u,v) \wedge \neg(V_1(u) \wedge V_2(u))] \wedge \\ [\forall u.B(u) \leftrightarrow V_1(u) \wedge \neg V_2(u)] \text{ mit} \\ \phi_{\subseteq}(X,Y) = \forall x.X(x) \rightarrow Y(x).\end{aligned}$$

Durch diese Formel sind die Mengen $A$ und $B$ für eine Menge von gewählten und durch $S$ repräsentierte Sterne eindeutig bestimmt. Die Summe der Einträge für Indizes $(i_1,i_2,i_3)$ mit fest gewählter Komponente $i_1$ und $i_2 \leq i_3$ des Histogramms gibt somit die Anzahl der $i_1$-elementigen Teilmengen der Sterne an, die die durch die obige Ungleichung geforderte Eigenschaft besitzen. Die Summe aller Einträge $(i_1,i_2,i_3)$ mit $i_2 \leq i_3$ des Lösungshistogramms entspricht also der Anzahl von möglichen Teilmengen von Sternen mit der geforderten Eigenschaft. ∎

Auch für dieses Problem findet sich eine schwach-NP-vollständige Variante bei Garey und Johnson (19): RANDOMIZATION-TEST-FOR-MATCHED-PAIRS. Bei diesem Entscheidungsproblem soll für ein gegebenes $k$ berechnet werden ob $k$ oder mehr Teilmengen $S$ der gegebenen Tupel existieren, sodass

$$\sum_{i\in S}|x_i-y_i| \leq \sum_{\substack{x_i>y_i,\\ i\in S}}(x_i-y_i)$$

für diese Mengen gilt. Für dieses Problem ergibt sich mit dem obigen Satz das folgende Korollar.

KOROLLAR 4.21.

$p_{\max}$-*RANDOMIZATION-TEST-FOR-MATCHED-PAIRS* $\in$ PLS. ∎

## 4.2 SCHEDULINGPROBLEME

Als Schedulingprobleme werden Probleme bezeichnet, bei denen Ressourcen zeitlich aufgeteilt werden, um eine Menge von Aufgaben zu bearbeiten (9). Diese Ressourcen können unterschiedlicher Art sein, sodass sich viele Probleme der Praxis als Scheduling-Probleme modellieren lassen. Das Spektrum der Ressourcen reicht hierbei von menschlicher Arbeitskraft und Geld über Energie bis zu Rechenzeit auf Computersystemen. Die gegebenen Aufgaben können dabei einer Struktur unterliegen, die die möglichen Bearbeitungsreihenfolgen der Aufgaben erheblich beeinflusst. Beispielsweise kann es notwendig sein, dass eine Aufgabe $a$ vor einer Aufgabe $b$ bearbeitet werden muss, da zur Bearbeitung von $b$ ein Ergebnis aus der Bearbeitung von $a$ notwendig ist. In dieser Arbeit liegt der Fokus auf dem einfachen Sequencing-Problem SEQUENCING, welches in seiner parametrisierten Version betrachtet wird.

- $p_{\max}$-SEQUENCING.
  - INSTANZ. Eine Menge von Aufgaben mit jeweils einer Start-, Lauf- und Endzeit.
  - PARAMETER. Der Wert der größten in der Eingabe kodierten Zahl.
  - FRAGE. Existiert für die gegebenen Aufgaben ein gültiger sequentieller Ablaufplan für einen Prozessor? Ein Ablaufplan ist genau dann gültig, wenn der Prozessor die Aufgaben so abarbeiten kann, dass
    1. keine Bearbeitung einer Aufgabe unterbrochen wird und
    2. die Bearbeitung einer Aufgabe frühestens mit der Startzeit beginnt und vor der Endzeit abgeschlossen ist.

Das Problem SEQUENCING ist stark-NP-vollständig. Eine schwach-NP-vollständige Variante ist $p_{\max}$-FIXED-RD-SEQUENCING, bei der die Anzahl der Start- und Endzeiten durch eine Konstante beschränkt ist (19). Diese Variante wird im Folgenden weiter untersucht.

Um eine gegebene Instanz von $p_{\max}$-FIXED-RD-SEQUENCING mittels MSO-Formeln auszuwerten, wird die folgende *Sequencing-Struktur* eingeführt. Es seien $v_1, \ldots, v_m$ die aufsteigend sortierten, paarweise unterschiedlichen Start- und Endzeiten, die in der Eingabe kodiert sind. Für jeden dieser Zeitpunkte wird ein Knoten konstruiert und diese *Zeitpunkt-Knoten* werden in aufsteigender Reihenfolge miteinander verbunden, sodass ein Pfad entsteht. Die Zeitpunktknoten werden durch die Relation $RD^S$ gefärbt. Für jede Aufgabe wird ein Stern konstruiert, dessen Wert der Länge der repräsentierten Aufgabe entspricht, und diese *Aufgaben-Sterne* werden mit ihren korrespondierenden Zeitpunkt-Knoten verbunden. Die Verbindungsknoten werden mit der Relation $Tasks^S$ gefärbt. In Abbildung 4.5 ist ein Beispiel einer solchen *Sequencing-Struktur* dargestellt.

Diese Konstruktion ist durch eine FPLS-Turingmaschine berechenbar, da für die

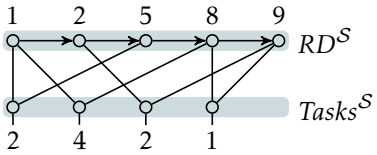



ABBILDUNG 4.5. Die obige Grafik stellt die Sequencing-Struktur für eine Instanz mir den Startzeiten 1, 2, 8, den Endzeiten 5, 8, 9 und den Aufgaben $\{(1,2,5),(1,4,8),(2,2,9),(8,1,9)\}$ dar. Die Laufzeiten der jeweiligen Aufgaben sind zugunsten der Übersichtlichkeit als Zahlen dargestellt.

Konstruktion der Sterne und zur Sortierung der konstant vielen Start- und Endzeitpunkte nur Zähler in $O(\log|x| + \log\kappa(x))$ benötigt werden. Weiter hat diese logische Struktur eine konstante Baumweite.

LEMMA 4.22. *Die Sequencing-Struktur hat für $k > 3$ Start- und Endzeiten eine Baumweite von $k-1$.*

BEWEIS. Eine entsprechende Baumzerlegung fasst die $k$ Zeitpunkt-Knoten zu einem Bag der Größe $k$ zusammen, die mit Bags der Größe 3 für Verbindungs-Knoten der Aufgaben-Sterne und ihren zugehörigen Zeitpunkt-Knoten verbunden sind. Mit letzteren sind dann Bags der Größe 2 für die Wertknoten der Aufgaben-Sterne verbunden. Die Baumweite dieser Baumzerlegung ist somit durch $k-1$ beschränkt.

Durch die Sequencing-Struktur kann dann $p_{\max}$-FIXED-RD-SEQUENCING entschieden werden.

SATZ 4.23.

$p_{\max}$-*FIXED-RD-SEQUENCING* $\in$ PLS.

BEWEIS. Aus einer gegebenen Instanz für $p$-FIXED-RD-SEQUENCING wird zunächst eine Sequencing-Struktur konstruiert. Durch eine MSO-Formel $\phi_{\text{FIXED-RD-SEQUENCING}}$ mit freien Variablen $I_1,\ldots,I_{m-1}$ zweiter Stufe, im Folgenden als Zeitintervalle interpretiert, werden die folgenden Bedingungen ausgedrückt:

1. Die Variablen $I_i$ partitionieren die Menge der Wertknoten der Aufgaben-Sterne.
2. Jedes Intervall $I_i$ beinhaltet nur Wertknoten von Aufgaben-Sternen, deren Startzeit vor $v_i$ liegt oder $v_i$ ist.
3. Jedes Intervall $I_i$ beinhaltet nur Wertknoten von Aufgaben-Sternen, deren Endzeit hinter $v_{i+1}$ liegt oder $v_{i+1}$ ist.
4. Für alle zwei *benachbarte Intervalle* $I_i$ und $I_{i+1}$ gilt, dass keine zwei unterschiedlichen Aufgaben-Sterne existieren, von denen jeweils Wert-Knoten in zugleich in beiden Intervallen sind.

5. Für alle drei Intervalle $I_{i_1}$, $I_{i_2}$ und $I_{i_3}$ mit $i_1 < i_2 < i_3$ gilt, dass wenn $I_{i_1}$ und $I_{i_3}$ Wertknoten eines gemeinsamen Aufgaben-Sterns beinhalten, das Intervall $I_{i_2}$ ausschließlich Wertknoten dieses Aufgaben-Sterns beinhaltet.

Da durch die konstante Anzahl von Start- und Release-Zeiten auch die Anzahl der Intervalle $I_1, \ldots, I_{m-1}$ konstant ist, sind die obigen Bedingungen innerhalb der MSO-Formel durch Fallunterscheidungen ausdrückbar.

Das Histogramm von $\phi_{\text{FIXED-RD-SEQUENCING}}$ und der gegebenen Struktur wird nun auf einen echt positiven Eintrag an einem Index $(i_1, \ldots, i_{m-1})$ mit $i_j \leq v_{j+1} - v_j$ für $j \in \{1, \ldots, m-1\}$ untersucht, wobei $v_j$ wie oben den $j$-ten Zeitpunkt in der aufsteigend sortierten Folge von in der Eingabe kodierten Start- und Endzeiten bezeichnet. Ist ein solcher Eintrag vorhanden, existiert ein gültiger Ablaufplan.

Zu beachten ist, dass die obige Formel für eine Belegung ihrer freien Variablen unter Beachtung des Histogramms wahr sein kann, ohne dass diese Belegung einem gültigen Ablaufplan entspricht, da die letzte der obigen Bedingungen nicht fordert, dass in dem Intervall $I_{i_2}$ keine Unterbrechung einer Aufgabenbearbeitung stattfindet. Jedoch impliziert eine solche gültige Belegung der Formel, dass ein gültiger Ablaufplan existiert, da eine Unterbrechung der Bearbeitung einer Aufgabe ausgeglichen werden kann, indem Wertknoten des betreffenden Aufgaben-Sterns in das entsprechende Intervall verlegt werden können, ohne dass die Formel ihre Gültigkeit verliert. ∎

## 4.3 GRAPHENPROBLEME

Graphenprobleme haben viele Anwendungsbereiche. Das Spektrum reicht hier von Physik, Chemie und Biologie über Soziologie und Ökonomie bis zur Linguistik. Graphen können für nahezu alle Anwendungsbereiche verwendet werden, in denen eine binäre Relation verwendet wird. Weiter ermöglicht ihre einfache Darstellung ein schnelles und intuitives Verständnis für die präsentierten Sachverhalte. Daher ist die Untersuchung von Graphenproblemen von besonderer Wichtigkeit (21).

Viele Instanzen der im Folgenden betrachteten Probleme bestehen aus einem Graphen und einer natürlichen Zahl. Diese können zusammen in einer logischen Struktur $\mathcal{S}$ kodiert werden, indem die für alle Strukturen geforderte Ordnungsrelation $\leq^{\mathcal{S}}$ verwendet wird, um die natürlichen Zahlen als binäre Strings zu kodieren. Diese Kodierung ist möglich, da sie die Baumweite der Struktur nicht erhöht und leicht zu berechnen ist, wie in den Abschnitten 2.2 und 2.3 gezeigt wurde. Zugunsten der besseren Lesbarkeit werden im Folgenden diese Konzepte von Graphen und Zeichenketten getrennt betrachtet, obwohl sie in einer gemeinsamen logischen Struktur kodiert werden.

Viele der folgenden Graphenprobleme werden über die Baumweite parametrisiert. Diese Probleme sind damit keine parametrisierten Probleme nach Definition 3.1,

da die Parametrisierung nicht FL-berechenbar, sondern nur FXL-berechenbar ist. Probleme dieser Art werden im Folgenden mit dem Präfix $p_{\text{tw}}$- versehen.

Ein wichtiges Graphenproblem ist CLIQUE, das Finden von Cliquen einer gegebenen Größe. Cliquen sind Graphen, in denen jedes Knotenpaar durch eine Kante direkt miteinander verbunden ist. Der Cliquen-Begriff wurde von Luce und Perry (32) bei der Modellierung von sozialen Cliquen in den Sozialwissenschaften geprägt. Ein erster Algorithmus zur Berechnung von Cliquen stammt von Harary und Ross (22). Karp (30) zeigte, dass das Cliquenproblem NP-vollständig ist. Im Folgenden wird das parametrisierte Cliquen-Problem in einer Zählvariante untersucht.

- $p_{\text{tw}}$-COUNT-CLIQUES. Für einen Graphen soll berechnet werden, wie viele Knotenmengen der Größe $l$ existieren, sodass jeder Knoten der Menge mit jedem anderen Knoten der Menge über eine direkte Kante verbunden ist.
  - INSTANZ. Ein Graph $\mathcal{G}$ und eine natürliche Zahl $l$.
  - PARAMETER. Die Baumweite von $\mathcal{G}$.
  - ERGEBNIS. Die Anzahl der Cliquen der Größe $l$ von $\mathcal{G}$.

- SATZ 4.24.

  $p_{\text{tw}}$-*COUNT-CLIQUES* $\in$ FXL. ∎

- BEWEIS. Die MSO-Formel

$$\phi_{\text{CLIQUE}}(C) = \forall uv.C(u) \wedge C(v) \rightarrow E(u,v)$$

drückt aus, dass $C$ eine Clique ist. Die Anzahl der Cliquen der Größe $l$ des Graphen $\mathcal{G}$ kann dann aus dem Histogramm am Index $l$ abgelesen werden. Das Histogramm ist nach Satz 2.28 FXL-berechenbar. ∎

Ein weiteres bekanntes Graphenproblem ist VERTEX-COVER, das Berechnen einer Knotenüberdeckung mit einer gegebenen maximalen Größe. Eine Knotenüberdeckung ist eine Menge von Knoten, sodass jede Kante des gegebenen Graphen mit einem Knoten der Knotenüberdeckung inzident ist. Dieses Problem gehört zu den ersten Problemen, für die gezeigt wurde, dass sie NP-vollständig sind (30) und wurde aufgrund seiner vielfältigen Anwendungsbereiche, beispielsweise in der Biologie und der Bioinformatik, bis heute intensiv untersucht. In dieser Arbeit wird das parametrisierte Knotenüberdeckungsproblem betrachtet, welches über die Baumweite des gegebenen Graphen parametrisiert wird. Diese Parametrisierung unterscheidet sich von der oft gewählten Parametrisierung, bei der die gegebene maximale Größe gewählt wird (11).

- $p_{\text{tw}}$-COUNT-VERTEX-COVERS. Für einen Graphen soll berechnet werden, wie viele

Knotenmengen der Größe $l$ existieren, sodass jede Kante mit einem Knoten dieser Menge inzident ist.

- INSTANZ. Ein Graph $\mathcal{G}$ und eine natürliche Zahl $l$.
- PARAMETER. Die Baumweite von $\mathcal{G}$.
- ERGEBNIS. Die Anzahl der Knotenüberdeckungen der Größe $l$.

SATZ 4.25.

$p_{\text{tw}}$-*COUNT-VERTEX-COVERS* $\in$ FXL. ∎

BEWEIS. Der Beweis entspricht dem von Satz 4.24 mit der MSO-Formel

$$\phi_{\text{VERTEX-COVER}}(C) = \forall uv.E(u,v) \rightarrow C(u) \vee C(v).$$ ∎

Für die Minimierungsvariante $p_{\text{tw}}$-MINIMUM-VERTEX-COVER, bei der nach der Größe einer kleinstmöglichen Knotenüberdeckung gefragt wird, ergibt sich aus diesem Satz das folgende Korollar, indem nach dem kleinsten Index mit einem echt positiven Eintrag im Histogramm gesucht wird.

KOROLLAR 4.26.

$p_{\text{tw}}$-*MINIMUM-VERTEX-COVER* $\in$ FXL. ∎

Das Problem DOMINATING-SET, in einem Graphen eine Knotenmenge von gegebener Größe zu finden, sodass jeder Knoten entweder ein Knoten dieser Menge ist oder mit einem Knoten dieser Menge über eine direkte Kante verbunden ist, ist ein wichtiges Problem im Bereich der verteilten Systeme (43, 29, 1). Dominierende Mengen können genutzt werden, um in einem Netzwerk eine möglichst kleine Menge von Stationen zu berechnen, sodass jeder Knoten des Netzwerks eine direkte Verbindung zu einer solchen Station hat. Ein Beispielszenario findet sich im Bereich der Sensornetzwerke: Für ein gegebenes Sensornetzwerk soll eine Menge von Sensorknoten ermittelt werden, die mit speziellen Zusatzfunktionen versehen werden, sodass jeder Knoten eine direkte Verbindung mit einem solchen Spezialknoten hat und dass die Menge dieser Spezialknoten möglichst klein ist, um die zusätzlichen Kosten durch die Einrichtung der Spezialknoten gering zu halten.

- $p_{\text{tw}}$-COUNT-DOMINATING-SETS. Für einen Graphen soll berechnet werden, wie viele dominierende Mengen der Größe $l$ existieren, also wie viele Mengen von $l$ Knoten es gibt, sodass jede Kante des Graphen mit einem Knoten dieser Menge inzident ist.
  - INSTANZ. Ein Graph $\mathcal{G}$ und natürliche Zahl $l$.
  - PARAMETER. Die Baumweite von $\mathcal{G}$.
  - ERGEBNIS. Die Anzahl von dominierenden Mengen der Größe $l$.

- SATZ 4.27.

  $p_{\text{tw}}$-*COUNT-DOMINATING-SETS* $\in$ FXL. ■

- BEWEIS. Der Beweis entspricht dem von Satz 4.24 mit der MSO-Formel

  $\phi_{\text{DOMINATING-SET}}(C) = \forall v . C(v) \vee \exists w . C(w) \wedge E(v, w)$. ■

Auch für dieses Problem ergibt sich dann, dass die Minimierungsvariante, bei der die Größe der kleinsten dominierenden Menge gesucht wird, FXL-berechenbar ist.

- KOROLLAR 4.28.

  $p_{\text{tw}}$-*MINIMUM-DOMINATING-SET* $\in$ FXL. ■

Zu den bekanntesten Graphenproblemen gehören Knotenfärbungsprobleme, bei denen nach der Anzahl von Farben gefragt wird, die benötigt werden, um die Knoten eines Graphen so zu färben, dass keine zwei Knoten, die direkt miteinander verbunden sind, die gleiche Farbe haben. Färbungsprobleme haben viele Anwendungen, beispielsweise dienen sie zum Aufbrechen von Symmetrien in der Parallelverarbeitung. Parallele Algorithmen benötigen oftmals Unabhängigkeiten zwischen Datenobjekten, um korrekt arbeiten zu können. Diese Unabhängigkeiten können durch Färbungen ermittelt werden, indem die Datenobjekte als Knoten und die Abhängigkeiten als Kanten modelliert werden. In einer gültigen Färbung eines solchen Graphen sind dann gleich gefärbte Objekte voneinander unabhängig. Durch eine Färbung mit wenigen Farben lassen sich so große unabhängige Mengen finden und Symmetrien aufbrechen. Erste komplexitätstheoretische Betrachtungen dieses Problems stammen von Karp (30). Motiviert durch Probleme wie Symmetriebrechung (40), Scheduling oder Register-Allokation (10, 8) finden sich viele Untersuchungen zu Färbungen.

- $p_{\text{tw}}$-CHROMATIC-NUMBER. Für einen Graphen soll die chromatische Zahl berechnet werden. Die chromatische Zahl ist dabei die kleinste Zahl $c$, sodass die Knoten des Graphen mit $c$ Farben gefärbt werden können, also jeder Knoten genau eine Farbe erhält und keine zwei Knoten, die durch eine Kante miteinander verbunden sind, die gleiche Farbe haben.
  - INSTANZ. Ein Graph $\mathcal{G}$.
  - PARAMETER. Die Baumweite von $\mathcal{G}$.
  - ERGEBNIS. Die minimale Anzahl der für eine gültige Färbung von $\mathcal{G}$ benötigten Farben.

- SATZ 4.29.

  $p_{\text{tw}}$-*CHROMATIC-NUMBER* $\in$ FXL. ■

■ BEWEIS. Bisher ist keine MSO-Formel bekannt, mit der für ein beliebiges $c$ die $c$-Färbbarkeit eines Graphen ausgedrückt werden kann. Dennoch ist es möglich, die chromatische Zahl auf logarithmischem Platz zu berechnen. Hierfür wird die Eigenschaft verwendet, dass die chromatische Zahl eines Graphen mit Baumweite $k$ höchstens $k+1$ sein kann (18). Durch sukzessives Prüfen, ob der gegebene Graph $c$-färbbar für $c = 1,\ldots,k+1$ ist, kann die chromatische Zahl berechet werden.

Dazu sei das Problem $p_{\text{tw}}$-$c$-COLORABILITY wie folgend definiert.

- INSTANZ. Ein Graph $\mathcal{G}$ und eine natürliche Zahl $c$.
- PARAMETER. Die Baumweite von $\mathcal{G}$.
- FRAGE. Existiert eine gültige Färbung von $\mathcal{G}$ mit $c$ Farben?

Es gilt $p_{\text{tw}}$-$c$-COLORABILITY $\in$ XL für jedes $c$, da für jedes feste $c$ durch eine MSO-Formel ausgedrückt werden kann, dass eine gültige $c$-Färbung existiert. Die Formel für $c = 3$ ist beispielsweise

$$\begin{aligned}\phi_{3\text{-COLORABILITY}}(R,G,B) = \forall v.\,&[R(v) \vee B(v) \vee G(v)] \wedge \\ &[\neg(R(v) \wedge G(v)) \wedge \\ &\neg(G(v) \wedge B(v)) \wedge \\ &\neg(R(v) \wedge B(v))] \wedge \\ \forall uv.\,E(u,v) \rightarrow\ &\neg(R(u) \wedge R(v)) \wedge \\ &\neg(G(u) \wedge G(v)) \wedge \\ &\neg(B(u) \wedge B(v)).\end{aligned}$$

Existiert ein positiver Eintrag im Histogramm einer Formel $\phi_{c\text{-COLORABILITY}}$, so existiert eine $c$-Färbung. Dieses Histogramm kann nach dem Satz von Courcelle unter Einhaltung der entsprechenden Platzschranken ausgewertet werden.

Die Turingmaschine für $p_{\text{tw}}$-CHROMATIC-NUMBER entscheidet nun sukzessive für $c = 1,\ldots,k+1$ das Problem $p_{\text{tw}}$-$c$-COLORABILITY für den gegebenen Graphen, wobei $k$ die Baumweite des gegebenen Graphen bezeichnet. ■

## 4.4 ERFÜLLBARKEITSPROBLEME

Eines der zentralen Probleme der Komplexitätstheorie ist das Erfüllbarkeitsproblem der Aussagenlogik SATISFIABILITY, bei der nach der Existenz einer erfüllenden Belegung für eine gegebene aussagenlogische Formel gefragt wird. SATISFIABILITY war das erste als NP-vollständig identifizierte Problem (13) und bildete den Anfang einer Vielzahl von Untersuchungen zur NP-Vollständigkeit von Problemen (30, 19). In diesem Abschnitt wird die über die Variablenvorkommnisse parametrisierte Variante $p_{\text{var}}$-SATISFIABILITY hinsichtlich ihrer Platzkomplexität untersucht und gezeigt, dass dieses Problem in der Klasse XL ist.

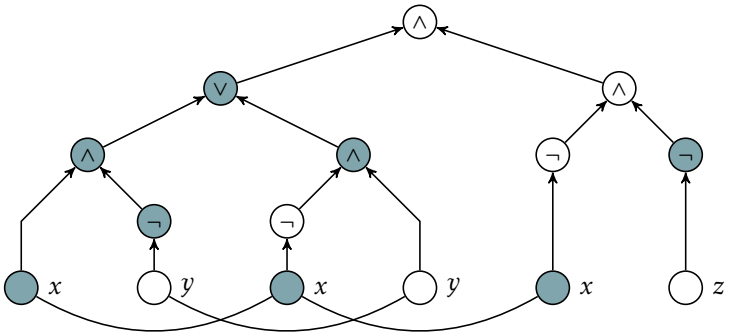
Aussagenlogische Formeln lassen sich mit dem Konzept der Baumweite in Verbindung bringen, indem sie als Graphen modelliert werden. Hierfür wird an dieser

Stelle ein erweiterter Syntaxbaum eingeführt, der das Konzept des Syntaxbaums für aussagenlogische Formeln ergänzt.

■ DEFINITION 4.30. Ein *Formel-Graph* ist ein Graph $\mathcal{G} = (V, E^{\mathcal{S}}, R_{\wedge}^{\mathcal{S}}, R_{\vee}^{\mathcal{S}}, R_{\neg}^{\mathcal{S}}, r^{\mathcal{S}})$ mit einstelligen Relationen $R_{\wedge}^{\mathcal{S}}$, $R_{\vee}^{\mathcal{S}}$, $R_{\neg}^{\mathcal{S}}$ und der Konstante $r^{\mathcal{S}}$, sodass

1. $\mathcal{G}$ aus einem gerichteten Baum mit Wurzel $r^{\mathcal{S}}$ besteht, wobei jede des Baumes Kante in Richtung der Wurzel zeigt und die Blattknoten durch ungerichtete Kanten zu knotendisjunkten Pfaden verbunden sein dürfen,
2. die Mengen $R_{\wedge}^{\mathcal{S}}$, $R_{\vee}^{\mathcal{S}}$ und $R_{\neg}^{\mathcal{S}}$ eine Partitionierung der inneren Knoten des Graphen bilden,
3. die Blätter des Baumes keine Elemente der Mengen $R_{\wedge}^{\mathcal{S}}$, $R_{\vee}^{\mathcal{S}}$ und $R_{\neg}^{\mathcal{S}}$ sind,
4. jeder Knoten der Relation $R_{\neg}^{\mathcal{S}}$ genau ein Kind hat,
5. jeder Knoten der Relation $R_{\wedge}^{\mathcal{S}}$ genau zwei Kinder hat. ■

Ein solcher Baum korrespondiert zu einer aussagenlogischen Formel, wie Abbildung 4.6 zeigt. Die Blätter des Baumes entsprechen den Vorkommnissen der freien Variablen und die inneren Knoten den Verknüpfungen der Formel, wobei durch die Relationen $R_{\wedge}^{\mathcal{S}}$, $R_{\vee}^{\mathcal{S}}$ und $R_{\neg}^{\mathcal{S}}$ jedem inneren Knoten eine Verknüpfungsoperation zugewiesen ist. Variablenvorkommnisse gleicher Variablen sind miteinander durch Kanten verbunden und bilden so knotendisjunkte Pfade.




■ ABBILDUNG 4.6. Der zu der Formel $\varphi = ((x \wedge \neg y) \vee (\neg x \wedge y)) \wedge (\neg x \wedge \neg z)$ korrespondierende Graph. Die inneren Knoten sind entsprechend ihrer Verknüpfungsoperation beschriftet. Die Blätter, die Vorkommnissen gleicher Variablen entsprechen, sind in Form von induzierten knotendisjunkten Pfaden miteinander durch ungerichtete Kanten verbunden. Weiter hat der Graph eine Färbung, die zu einer Anfangsfärbung korrespondiert, bei der die Knoten gefärbt sind, die zu den Variablenvorkommnissen der Variable $x$ gehören. Dies entspricht der Belegung, bei der genau die Variable $x$ wahr ist. Da der Wurzelknoten nicht gefärbt ist, handelt es sich nicht um eine erfüllende Belegung.

Zunächst wird eine Variante der parametrisierten Zählversion des Erfüllbarkeitsproblems für aussagenlogische Formeln betrachtet.

- $p_{\text{tw}}$-COUNT-SATISFIABILITY.
  - INSTANZ. Eine aussagenlogische Formel $\varphi$.
  - PARAMETER. Die Baumweite des Formel-Graphen $\mathcal{G}$ von $\varphi$.
  - ERGEBNIS. Die Anzahl der erfüllenden Belegungen von $\varphi$.
- SATZ 4.31.

  $p_{\text{tw}}$-*SATISFIABILITY* $\in$ FXL. ∎

- BEWEIS. Zunächst wird aus der aussagenlogischen Formel $\varphi$ der korrespondierende Formel-Graph berechnet. Dessen Größe ist polynomiell in der Länge von $\varphi$ und von einer logarithmisch platzbeschränkten Turingmaschine berechenbar.

Jeder Knoten des Graphen entspricht einer Teilformel von $\varphi$. Gesucht ist nun eine gültige Färbung der Knoten, sodass ein Knoten genau dann gefärbt ist, wenn seine korrespondierende Teilformel wahr ist. Dies lässt sich durch eine logische Formel $\phi_{\text{SAT}}$ mit der freien Variablen $T$ ausdrücken, wobei $T$ genau die gefärbten Blattknoten enthält, also den auf Wahr gesetzten Variablenvorkommnissen von $\varphi$ entspricht. Die Formel $\phi_{\text{SAT}}$ ist genau dann wahr, wenn diese Ausgangsfärbung widerspruchsfrei ist und zu einer gültigen Gesamtfärbung $S$ erweitert werden kann, die den Wurzelknoten färbt.

Die Ausgangsfärbung ist genau dann widerspruchsfrei, wenn für jeden Pfad innerhalb der Blattknoten gilt, dass entweder alle oder kein Knoten des Pfades gefärbt ist, also die Vorkommnisse der gleichen Variablen in der Formel eine gleiche Belegung haben. Dies wird durch die Formel $\varphi_{=}$ ausgedrückt.

Eine gültige Gesamtfärbung liegt vor, wenn die folgenden Eigenschaften erfüllt sind:

1. Ein Knoten $(\neg)$ ist genau dann gefärbt, wenn sein Kind nicht gefärbt ist.
2. Ein Knoten $(\wedge)$ ist genau dann gefärbt, wenn alle seine Kinder gefärbt sind.
3. Ein Knoten $(\vee)$ ist genau dann gefärbt, wenn mindestens eines seiner Kinder gefärbt ist.

Eine zu einer Anfangsfärbung passenden Gesamtfärbung zeigt Abbildung 4.6.

Die obigen Eigenschaften werden durch die logischen Formeln $\phi_{\neg}$, $\phi_{\wedge}$ und $\phi_{\vee}$ ausgedrückt. Die Formel $\phi_L$ ist für einen Knoten $v$ genau dann wahr, wenn $v$ ein Blattknoten ist. Damit ergibt sich die logische Formel $\phi_{\text{SAT}}$ zu

$$\begin{aligned}
\phi_{\text{SAT}}(T) &= \varphi_{\text{LEAFS}}(T) \wedge \exists S \,.\, \phi_{\subseteq}(T,S) \wedge S(r) \wedge \\
&\qquad \varphi_{=}(S) \wedge \varphi_{\wedge}(S) \wedge \varphi_{\vee}(S) \wedge \varphi_{\neg}(S) \text{ mit} \\
\varphi_{\text{LEAFS}}(T) &= \forall u \,.\, T(u) \to \varphi_L(u), \\
\varphi_{=}(S) &= \forall u v \,.\, \phi_{\varphi\text{-PATH}}(u,v) \to (T(u) \leftrightarrow T(v)), \\
\varphi(u_\varphi, v_\varphi) &= E(u_\varphi, v_\varphi) \wedge \varphi_L(u_\varphi) \wedge \varphi_L(v_\varphi),
\end{aligned}$$

$$\varphi_\wedge(S) = \forall u . R_\wedge(u) \to [S(u) \leftrightarrow (\forall v . E(v,u) \to S(v))],$$
$$\varphi_\vee(S) = \forall u . R_\vee(u) \to [S(u) \leftrightarrow \exists v . E(v,u) \wedge S(v)],$$
$$\varphi_\neg(S) = \forall u . R_\neg(u) \to [S(u) \leftrightarrow \forall v . E(v,u) \wedge \neg S(v)],$$
$$\varphi_L(u) = \neg(R_\wedge(u) \vee R_\vee(u) \vee R_\neg(u)).$$

Aus dem Histogramm von $\phi_{\text{SAT}}$ kann abgelesen werden, wie viele erfüllende Belegungen existieren: An der Stelle $l$ ist die Anzahl der erfüllenden Belegungen mit $l$ auf Wahr gesetzten Variablenvorkomnissen ablesbar. Die Summe der Einträge des Histogramms ergibt gerade die Zahl aller erfüllenden Belegungen. ■

Für das Entscheidungsproblem $p_{\text{tw}}$-SATISFIABILITY lässt sich mit diesem Satz sofort das folgende Korollar zeigen.

■ KOROLLAR 4.32.

*$p_{\text{tw}}$-SATISFIABILITY* $\in$ XL. ■

Bei den betrachteten Problemen handelt es sich nicht um parametrisierte Probleme im Sinne von Definition 3.1, da der Parameter nicht FL-berechenbar ist. Jedoch ist die Baumweite eines Formel-Graphen durch die Variablenvorkommnisse in der Formel beschränkt, wie im Folgenden gezeigt wird. Über diese Untersuchungen zu den Variablenvorkommnissen lässt sich das Problem dann parametrisieren.

Ein Formel-Graph ohne mehrfache Vorkommnisse von Variablen hat die Baumweite 1, da es sich um einen Baum handelt. Mehrfache Variablenvorkommnisse induzieren zusätzliche Kanten, die die Blattknoten des Formel-Graphen zu Pfaden verbinden. Hierdurch verliert der Formel-Graph seine Baumeigenschaft und die Baumweite erhöht sich. Bereits das zweifache Vorkommen jeder Variable kann dazu führen, dass die Baumweite der Formel-Graphen mit steigender Anzahl an Variablen ebenfalls unbeschränkt steigt. Abbildung 4.7 zeigt einen solchen Graphen.

Abbildung 4.7 lässt vermuten, dass die Baumweite eines Formel-Graphen mit der Verteilung der Vorkommnisse gleicher Variablen in der korrespondierenden Formel zusammenhängt: Wären die Variablenvorkommnisse so verteilt, dass die Kinder des Wurzelknotens keine gemeinsamen Variablen hätten, so würde der Formel-Graph keine Clique mit vier Knoten als Minor beinhalten. Tatsächlich lässt sich zeigen, dass ein Formel-Graph beschränkte Baumweite hat, wenn für alle Knoten gilt, dass seine Kinder beschränkt viele gemeinsame Variablen besitzen. Dies wird durch das folgende, allgemeinere Lemma impliziert, dessen Beweis jedoch noch eine Definition benötigt.

■ DEFINITION 4.33. Eine *Traversierungsfolge* eines Baumes ist eine Folge von Knoten, die folgende Eigenschaften erfüllt:

1. Der erste Knoten der Knotenfolge ist der Wurzelknoten des Baumes.

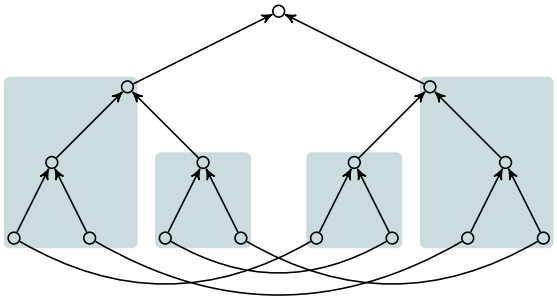


ABBILDUNG 4.7. Der dargestellte Graph kodiert eine aussagenlogische Formel mit vier freien Variablen, die jeweils zweimal vorkommen. In dem unterliegenden ungerichteten Graphen, der aus dem dargestellten Graphen entsteht, wenn jede gerichtete Kante durch eine ungerichtete Kante ersetzt wird, ergibt sich eine Clique aus vier Knoten als Minor. Dies ist erkennbar, wenn die durch den farbigen Hintergrund markierten Knoten durch Kantenkontraktion zu einem Knoten zusammengefasst werden und der Wurzelknoten mit seinen beiden zugehörigen Kanten entfernt wird.

2. Jeder Knoten des gegebenen Graphen ist genau einmal in der Knotenfolge vorhanden.
3. Für alle Knoten $v_i$ mit $i \in \{1, 2, \ldots, n-1\}$ der $n$-elementigen Knotenfolge gilt, dass $v_{i+1}$ ein Kind von $v_i$ ist oder ein Knoten $u$ existiert, der im gegebenen Baum ein Vorgänger von $v_i$ und ein direkter Vorgänger von $v_{i+1}$ ist.

Eine solche Traversierungsfolge entspricht der Knotenfolge, die zu einer Preorder-Tiefensuche in einem Baum korrespondiert. Abbildung 4.8 zeigt eine solche Traversierungsfolge. Wie nun gezeigt wird, haben Strukturen, die Bäume und eine korrespondierende Traversierungsfolge kodieren, eine beschränkte Baumweite.

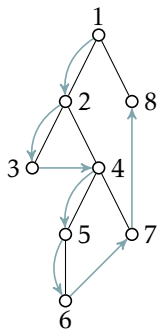


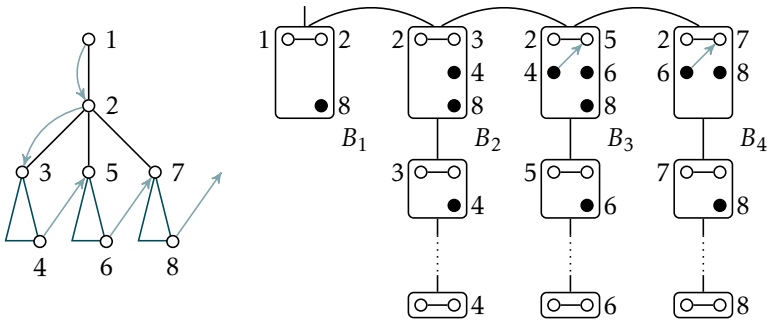

ABBILDUNG 4.8. Die obige Grafik zeigt einen ungerichteten Baum und eine durch die Kanten ↗ gekennzeichnete Traversierungsfolge $1, 2, \ldots, 8$. Die Wurzel ist der Knoten mit der Nummer 1.

■ LEMMA 4.34. *Wird eine Traversierungsfolge für eine Formel-Struktur $\mathcal{S}$ als zweistellige Relation*

$$T^{\mathcal{S}} = \{(u,v) \mid v \text{ ist ein direkter Nachfolger von } u\}$$

*der Struktur $\mathcal{S}$ kodiert, so erhöht sich die Baumweite der Struktur um* 3. ■

■ BEWEIS. Um dieses Lemma zu beweisen, wird aus der Baumzerlegung des gegebenen Graphen eine Baumzerlegung der gegebenen Struktur konstruiert, deren Baumweite durch eine Konstante beschränkt ist. Diese Konstruktion wird anhand der folgenden Abbildung erläutert.




Auf der linken Seite ist ein Baum durch die Knoten ○ und die Kanten ／ dargestellt. Bei den eingezeichneten Kanten handelt es sich um ungerichtete Kanten, die jedoch durch beliebig gerichtete Kanten ersetzt werden können. Die an den Knoten 3, 5 und 7 angehängten Dreiecke symbolisieren Teilbäume an diesen Knoten, deren Wurzeln die genannten Knoten sind. Die Kanten ↗ zeigen die betrachteten Kanten der gegebenen Traversierungsfolge $1,2,3,\ldots,4,5,\ldots,6,7,\ldots,8,\ldots$, deren Knoten zur einfacheren Darstellung mit natürlichen Zahlen beschriftet sind. Ziel ist nun, eine Baumzerlegung des gegebenen Graphen so zu erweitern, dass die Kanten ↗ der Knotenfolge durch die Bags der Baumzerlegung abgedeckt sind und die Baumweite dabei nur um eine additive Konstante steigt.

Auf der rechten Seite ist eine mögliche Baumzerlegung des gegebenen Graphen dargestellt. Werden ausschließlich die Knoten ○ und die Kanten ／ betrachtet, so entspricht die Darstellung einer minimalen Baumzerlegung des gegebenen Baumes. Die Bags $B_1$, $B_2$, $B_3$ und $B_4$ decken die Kanten der Verzweigung des Baumes ab und korrespondieren jeweils zur Wurzel eines Teilbaums, indem die Bags unter $B_2$, $B_3$ und $B_4$ die Kanten der Unterbäume der Knoten 3, 5 und 7 abdecken.

Vorwärtsgerichtete Kanten ↗ der Knotenfolge, also Kanten, die entsprechend der Tiefensuche in die Tiefe des Graphen zeigen, entsprechen Kanten, die bereits im gegebenen Baum vorhanden sind. Somit sind sie in einer Baumzerlegung des Graphen bereits abgedeckt. Es verbleiben die rückwärtsgerichteten Kanten ↗, die aus der Tiefensuche zurückkehren. Dies sind Kanten, die von einem in einer Tiefensuche zuletzt besuchten Knoten eines Teilbaumes auf einen zuerst besuchten

Knoten eines benachbarten Teilbaumes zeigen, wobei zwei Teilbäume benachbart sind, wenn sie einen gemeinsamen Wurzelknoten haben. In der Grafik sind dies innerhalb des betrachteten Teilbaumes die Knotenpaare 4, 5 und 6, 7. Der Nachfolger von Knoten 8 ist in einem Nachbarbaum des betrachteten Teilbaumes mit Wurzelknoten 1.

Um nun eine Rückwärtskante abzudecken, beispielsweise die Kante von Knoten 4 zu Knoten 5, wird zu allen Bags, die in der Baumzerlegung auf dem direkten Weg zu dem Bag mit Knoten 5 sind, der Knoten 4 hinzugefügt, hier als ● dargestellt. So wird in dem betrachteten Beispiel der Knoten 4 auf dem Weg über Bag $B_2$, der zu seinem Teilbaum korrespondiert, bis zum Bag $B_3$ eingefügt. Analog geschieht dies mit Knoten 6 und Knoten 8. Knoten 8 wird konsequenterweise auch in die Bags $B_1$, $B_2$, $B_3$, $B_4$ eingefügt, da der Knoten, auf den die Rückwärtskante von Knoten 8 aus zeigt, in einem Nachbarbaum des betrachteten Baumes mit Wurzel 1 liegt. Auf diese Weise kann jede Kante der Knotenfolge durch einen Bag abgedeckt werden und die Eigenschaften einer Baumzerlegung bleiben erhalten.

Die Baumweite der ursprünglichen Baumzerlegung ist 1. Jeder Bag, zu dem ein Teilbaum korrespondiert, beinhaltet zwei Knoten. Durch die obige Konstruktion erhält jeder Bag maximal 3 weitere Knoten: Einen Knoten durch eine Kante, die von einem Nachbarbaum in den betrachteten Teilbaum zeigt, einen Knoten von einer Kante, die aus dem betrachteten Teilbaum in einen Nachbarbaum zeigt und einen Knoten von einer Kante, die aus dem in der Tiefensuche zuletzt betrachteten Nachbarbaum über die gemeinsame Wurzel $w$ der benachbarten Teilbäume hinweg in einen Teilbaum zeigt, der nicht die Wurzel $w$ hat. ■

Um diese Beobachtungen für eine natürlichere Parametrisierung des Erfüllbarkeitsproblems zu verwenden, wird hier der Begriff der *gemeinsamen Variablenvorkommnisse* eingeführt.

■ DEFINITION 4.35. Die *gemeinsamen Variablenvorkommnisse* einer aussagenlogischen Formel $\phi \diamond \psi$ mit $\diamond \in \{\wedge, \vee\}$ ist die Menge der Variablen, die sowohl in $\phi$ als auch in $\psi$ vorkommen. Die *maximale Anzahl von gemeinsamen Variablenvorkommnissen* einer aussagenlogischen Formel $\phi$ ist die maximale Anzahl von gemeinsamen Variablen über allen für eine Formel definierten gemeinsamen Variablenvorkommnissen. ■

Mit diesen Beobachtungen lässt sich das parametrisierte Erfüllbarkeitsproblem $p_{\text{var}}$-COUNT-SATISFIABILITY definieren, bei dem die Variablenvorkommnisse in der gegebenen Formel den Parameter definieren:

- $p_{\text{var}}$-COUNT-SATISFIABILITY.
  - INSTANZ. Eine aussagenlogische Formel $\varphi$.
  - PARAMETER. Die maximale Anzahl von gemeinsamen Variablenvorkommnissen der Formel $\varphi$.
  - ERGEBNIS. Die Anzahl der erfüllenden Belegungen von $\varphi$.

SATZ 4.36.

$p_{\text{var}}$-*COUNT-SATISFIABILITY* $\in$ FXL. ■

BEWEIS. Die Baumweite des Formelgraphen einer Formel ist durch die maximale Anzahl der gemeinsamen Variablenvorkommnissen beschränkt. Es folgt mit Satz 4.31 die Behauptung. ■

Im Gegensatz zu der Parametrisierung über die Baumweite ist dieser Parameter FL-berechenbar, da zur Bestimmung der maximalen Anzahl von gemeinsamen Variablenvorkommnissen nur Zähler in $O(\log|x|)$ benötigt werden.

## 4.5 MENGENPROBLEME

Eine Klasse der von Karp untersuchten 21 Probleme sind Mengenprobleme (30). Bei diesen werden Familien von Mengen betrachtet und deren Eigenschaften untersucht.

Mengenfamilien sind durch logische Strukturen zunächst recht einfach zu kodieren: Das Universum besteht aus der Grundmenge und einstellige Relationen kodieren die gegebenen Mengenfamilien über der Grundmenge. Die Baumweite dieser Struktur ist 0 oder −1, falls das Universum kein Element beinhaltet, und somit beschränkt. Auf diese Weise ist es jedoch nur möglich, eine feste Anzahl von Mengen zu kodieren, da diese bereits durch die Signatur festgelegt sind. Sollen beliebige Mengenfamilien kodiert werden, so ist dies über einstellige Relationen nicht möglich. Abhilfe schafft die im Folgenden vorgestellte Kodierungsform.

Gegeben ist eine Struktur $\mathcal{S}$ mit der zweistelligen Relation

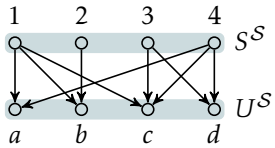
$$E^{\mathcal{S}} = \{(i,u) \mid u \in S_i\},$$

die die gegebenen Mengen kodiert, indem jedes Element, welches in mindestens einer der gegebenen Mengen vorkommt, mit einem *Verbindungselement* in Relation steht. Die Elemente der gegebenen Mengen werden in der einstelligen Relation $U^{\mathcal{S}}$ und die Verbindungselemente in der einstelligen Relation $S^{\mathcal{S}}$ zusammengefasst. Diese beiden Relationen sind disjunkt. Eine Familie $\{S_j\}$ von Mengen

$$S_1 = \{a,b,c\}$$
$$S_2 = \{b\},$$
$$S_3 = \{c,d\},$$
$$S_4 = \{a,c,d\}$$

wird somit als Struktur $\mathcal{S}$ kodiert, die als Graph in Abbildung 4.9 gezeigt wird. Eine solche Struktur wird im Folgenden auch direkt als *Mengenfamilie* bezeichnet.

Diese Struktur behebt das Problem der festen Anzahl von Mengen auf Kosten einer nicht unbedingt beschränkten Baumweite. Dies ist in Abbildung 4.9 zu




■ ABBILDUNG 4.9. Die Grafik zeigt die Mengenfamilie der Mengen $S_1 = \{a, b, c\}$, $S_2 = \{b\}$, $S_3 = \{c, d\}$, $S_4 = \{a, c, d\}$.

erkennen, die bereits für wenige Elemente eine hohe Vernetzung zeigt und eine ebenfalls hohe Baumweite vermuten lässt. Eine Aussage über die Baumweite dieser Strukturen macht der folgende Satz.

■ SATZ 4.37. *Sei $k$ die Anzahl der Elemente, die in mehreren Mengen einer Mengenfamilie vorkommen. Dann ist die Baumweite der Mengenfamilie durch $3k$ nach oben beschränkt.* ■

■ BEWEIS. Dieser Satz folgt aus Lemma 4.34 und dem zugehörigen Beweis, da eine Mengenfamilie ein Minor eines Formel-Graphen ist. Die Eingabe-Instanz entsteht aus dem Formel-Graphen, indem

1. die Wurzel entfernt,
2. für die inneren Knoten mehrere Kinder erlaubt und
3. die Pfade zwischen gemeinsamen Variablenvorkommnissen zu Knoten kontrahiert werden.

Diese Änderungen erhöhen die Baumweite des Graphen nicht. Jedes Element, welches in mehreren Mengen vorkommt, korrespondiert also zu einer Traversierungsfolge im Formel-Graphen. Nach Lemma 4.34 erhöht eine Traversierungsfolge die Baumweite der betrachteten Baumstruktur um 3. ■

Die Baumweite einer Mengenfamilie ist somit polynomiell in der Anzahl von gemeinsamen Elementen in den Mengen der Mengenfamilie. Diese Anzahl ist FPLS-berechenbar. In Anlehnung an die Parametrisierung über die Baumweite bei Graphenproblemen in Kapitel 4.3 werden die über die Anzahl von gemeinsamen Elementen in den Mengen der Mengenfamilie parametrisierten Probleme mit dem Präfix $p_{\text{mul}}$- versehen.

Das Mengenüberdeckungsproblem SET-COVERING gehört zu den bekanntesten kombinatorischen Mengen-Problemen der Theoretischen Informatik, da die Untersuchung dieses Problems zur Entwicklung von fundamentalen Techniken im Bereich der Approximationsalgorithmen geführt hat (42). An dieser Stelle wird zunächst die parametrisierte Zählvariante dieses Problems untersucht.

■ $p_{\text{mul}}$-COUNT-SET-COVERINGS. Für eine gegebene Familie $\{S_i\}$ von Mengen und eine natürliche Zahl $l$ soll berechnet werden, wie viele Teilfamilien von $l$ Mengen die

gegebene Mengenfamilie enthält, deren Vereinigung der Menge aller Elemente $U^{\mathcal{S}}$ der Familie entspricht.

- INSTANZ. Eine Mengenfamilie $\mathcal{S}$ und eine natürliche Zahl $l$.
- PARAMETER. Die Anzahl der Elemente, die in mehreren Mengen zugleich vorkommen.
- ERGEBNIS. Die Anzahl von Teilfamilien der Größe $l$, deren Vereinigung die Menge $U^{\mathcal{S}}$ ergibt.

SATZ 4.38.

$p_{\text{mul}}$-*COUNT-SET-COVERING* $\in$ FXL. ∎

BEWEIS. Die MSO-Formel

$$\phi_{\text{SET-COVERING}}(C) = \phi_{\subseteq}(C,S) \wedge (\forall u.U(u) \rightarrow \exists v.E(v,u) \wedge C(v)).$$

drückt aus, dass $C$ eine Teilmenge der Verbindungselemente und somit eine Auswahl an Mengen ist, sodass jedes Element aus $U^{\mathcal{S}}$ mit mindestens einem der gewählten Verbindungselemente verbunden, also in mindestens einer der gewählten Mengen ist. Der Eintrag im Histogramm für den Index $l$ gibt an, wie viele Auswahlen von $l$ Mengen existieren, die alle Elemente von $U^{\mathcal{S}}$ abdecken. ∎

Für das Entscheidungsproblem $p_{\text{mul}}$-SET-COVERING bei dem für eine gegebene natürliche Zahl $k$ entschieden werden soll, ob eine Mengenüberdeckung der Größe $k$ existiert ergibt sich damit sofort das folgende Korollar.

KOROLLAR 4.39.

$p_{\text{mul}}$-*SET-COVERING* $\in$ XL. ∎

Über das Lösungshistogramm lässt sich auch die Minimierungsvariante berechnen, bei der nach der Größe der kleinsten Mengenüberdeckung gefragt wird.

KOROLLAR 4.40.

$p_{\text{mul}}$-*MINIMUM-SET-COVERING* $\in$ FXL. ∎

Eine Variante des Mengenüberdeckungsproblems ist EXACT-COVER, bei der exakte Mengenüberdeckungen gesucht werden. Wieder wird zunächst die parametrisierte Zählvariante betrachtet.

- $p_{\text{mul}}$-COUNT-EXACT-COVERS. Für eine gegebene Familie $\{S_i\}$ von Mengen und eine natürliche Zahl $l$ soll berechnet werden, wie viele Teilfamilien von $l$ paarweise

disjunkten Mengen die gegebene Mengenfamilie enthält, deren Vereinigung der Menge aller Elemente $U^{\mathcal{S}}$ der Familie entspricht.

- INSTANZ. Eine Mengenfamilie $\mathcal{S}$ und eine natürliche Zahl $l$.
- PARAMETER. Die Anzahl der Elemente, die in mehreren Mengen zugleich vorkommen.
- ERGEBNIS. Die Anzahl von Teilfamilien der Größe $l$ mit paarweise disjunkten Mengen, deren Vereinigung die Menge $U^{\mathcal{S}}$ ergibt.

- SATZ 4.41.

  $p_{\text{mul}}$-*COUNT-EXACT-COVERS* $\in$ FXL. ■

- BEWEIS. Der Beweis ist analog zu dem von Satz 4.38 mit der MSO-Formel

$$\phi_{\text{EXACT-COVER}}(C) = \phi_{\subseteq}(C,S) \wedge \forall u.U(u) \rightarrow \exists v.(C(v) \wedge E(v,u) \wedge \neg\exists w.C(w) \wedge E(w,u) \wedge v \neq w).$$ ■

Die Komplexität des Entscheidungsproblems $p_{\text{mul}}$-EXACT-COVER lässt sich aus diesem Satz direkt ableiten.

- KOROLLAR 4.42.

  $p_{\text{mul}}$-*EXACT-COVER* $\in$ XL. ■

Auch die Minimierungsvariante lässt sich mit Satz 4.41 berechnen.

- KOROLLAR 4.43.

  $p_{\text{mul}}$-*MINIMUM-EXACT-COVER* $\in$ FXL. ■

Ein weiteres Mengenproblem ist SET-PACKING, welches ebenfalls zu Karps 21 Problemen gehört. Auch hier wird zunächst die parametrisierte Zählvariante betrachtet.

- $p_{\text{mul}}$-COUNT-SET-PACKINGS. Für eine gegebene Familie $\{S_i\}$ von Mengen und eine natürliche Zahl $l$ soll berechnet werden, wie viele Teilfamilien von $l$ paarweise disjunkte Mengen die gegebene Mengenfamilie enthält.
  - INSTANZ. Eine Mengenfamilie $\mathcal{S}$ und eine natürliche Zahl $l$.
  - PARAMETER. Die Anzahl der Elemente, die in mehreren Mengen zugleich vorkommen.
  - ERGEBNIS. Die Anzahl von Teilfamilien der Größe $l$, wobei die Mengen der Mengenfamilie paarweise disjunkt sind.

- SATZ 4.44.

  $p_{\text{mul}}$-*COUNT-SET-PACKINGS* $\in$ FXL. ■

■ BEWEIS. Der Beweis ist analog zu dem von Satz 4.38 mit der MSO-Formel

$$\phi_{\text{SET-PACKING}}(P) = \phi_{\subseteq}(P,S) \wedge \\ \forall uv.P(u) \wedge P(v) \wedge u \neq v \rightarrow \neg\exists w.E(u,w) \wedge E(v,w).$$ ■

Für die Komplexität des Entscheidungsproblems gilt dann das folgende Korollar.

■ KOROLLAR 4.45.

$p_{\text{mul}}$-*SET-PACKING* $\in$ XL. ■

Die zugehörige Maximierungsvariante lässt sich mit diesem Satz ebenfalls FXL-berechnen.

■ KOROLLAR 4.46.

$p_{\text{mul}}$-*MAXIMUM-SET-PACKING* $\in$ FXL. ■

# KAPITEL 5

## ZUSAMMENFASSUNG

In dieser Arbeit wurde die parametrisierte Komplexitätstheorie um ein Framework zur Klassifikation von Problemen bezüglich ihrer Platzkomplexität entwickelt und wurden anhand dieses Frameworks erste systematische Untersuchungen von Problemen durchgeführt. Diese Untersuchungen nutzten die Logspace-Varianten der Sätze von Bodlaender und Courcelle und zeigten verschiedene Techniken für deren Anwendung.

Die Berechnung von schwach-NP-vollständigen Zahlenproblemen mittels dynamischer Programmierung ist oft von inakzeptabel großem Speicherplatzbedarf begleitet, weshalb eine Untersuchung dieser Probleme bezüglich einer platzeffizienten Lösbarkeit notwendig ist. Das in dieser Arbeit vorgestellte Framework ergänzt die parametrisierte Komplexitätstheorie um die Platzklassen PLS, FPL und XL und eine hinreichend schwache FPLS-Reduktion, unter welcher die eingeführten Klassen abgeschlossen sind. Diese Klassen bilden eine zu der bekannten Zeitklassen-Hierarchie PPT ⊆ FPT ⊆ XP korrespondierende Platzklassen-Hierarchie PLS ⊆ FPL ⊆ XL und bieten damit erstmals eine Basis zur systematischen Untersuchung von Problemen bezüglich ihrer parametrisierten Platzkomplexität.

Das Spektrum der in dieser Arbeit untersuchten Probleme reicht von Zahlenproblemen über Scheduling- und Graphenprobleme bis zum Erfüllbarkeitsproblem für aussagenlogische Formeln und Mengenprobleme. Für diese Klassen von Problemen enthält diese Arbeit Untersuchungen zur Anwendbarkeit der Sätze von Bodlaender und Courcelle in ihren Logspace-Versionen, die sich hierbei als mächtige Werkzeuge erweisen.

Motiviert durch den hohen Speicherplatzbedarf von Algorithmen mit dynamischer Programmierung liegt der Schwerpunkt der in dieser Arbeit betrachteten Probleme auf Zahlenproblemen. Diese lassen sich zumeist über den größten in der Eingabe kodierten Wert parametrisieren und über eine Graphenkonstruktion mit der anschließenden Anwendung des Satzes von Courcelle in seiner Logspace-Version und das korrespondierende Histogramm lösen. Bei den in dieser Arbeit betrachteten Zahlenproblemen handelt es sich um klassische schwach-NP-vollständige Varianten von SUBSET-SUM, KNAPSACK, BIN-PACKING und INTEGER-PROGRAMMING. Auch das Scheduling-Problem für eine feste Anzahl von Start- und Endzeiten lässt sich über den größten in der Eingabe kodierten Wert parametrisieren und durch eine Graphenkonstruktion lösen. Probleme wie das Erfüllbarkeitsproblem SATISFIABILITY für aussagenlogische Formeln oder Mengenprobleme wie SET-COVERING oder EXACT-COVER korrespondieren auf natürliche

Weise zu Graphenproblemen und ihre Parametrisierung hat einen direkten Bezug zur Baumweite der betrachteten Graphen, wodurch diese Probleme der Klasse XL zugeordnet werden können.

## 5.1 AUSBLICK

Historisch bedingt beschäftigt sich die parametrisierte Komplexitätstheorie mit der Untersuchung von Problemen bezüglich ihrer Zeitkomplexität (16, 18). So sind die zentralen Klassen FPT und XP als zeitbeschränkte Klassen definiert. Auch andere Klassen wie die W-Hierarchie werden bezüglich ihrer Zeitkomplexität untersucht, indem für ihre Definition FPT-Reduktionen verwendet werden. Obwohl Downey und Fellows (16) bereits parametrisierte Platzklassen wie XL ansprechen und auch erste Untersuchungen in diese Richtung existieren (34), haben bisher keine systematischen Untersuchungen stattgefunden. Abbildung 5.1 stellt die angesprochenen Klassen zusammen mit den in dieser Arbeit eingeführen Klassen dar.

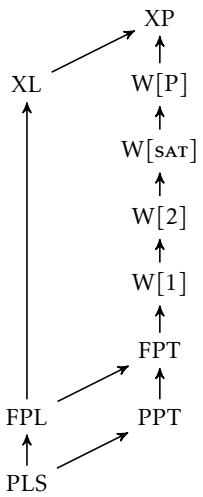



ABBILDUNG 5.1. Die obige Grafik zeigt die Einordnung der Klassen PLS, FPL und XL bezüglich der in der parametrisierten Komplexitätstheorie oft betrachteten Klassen FPT, XP und einigen Stufen der W-Hierarchie. Weiter ist die Klasse PPT der schwach-NP-vollständigen Probleme dargestellt.

Die Grafik zeigt, dass für die Stufen der W-Hierarchie keine korrespondierende Hierarchie zwischen den Platzklassen FPL und XL bekannt ist. Da die W-Hierarchie zur Klassifikation vieler parametrisierter Probleme verwendet wird, die sich von klassischen NP-vollständigen Problemen ableiten, ist es sinnvoll, eine korrespondierende Hierarchie von Platzklassen zu finden, die für die Untersuchung dieser Probleme bezüglich ihrer Platzkomplexität verwendet werden kann. Ähnlich wie die Sätze von Bodlaender und Courcelle könnten Meta-Theoreme für diese neuen

Klassen entwickelt und für eine erste komplexitätstheoretische Untersuchung von Problemen genutzt werden.

Die in dieser Arbeit präsentierten Untersuchungen zeigen, dass viele der durch dynamische Programmierung berechenbaren Probleme auch platzeffizient berechenbar sind. Allerdings können hierdurch keine Aussagen über die *simultane Platz- und Zeitkomplexität* von Problemen gemacht werden. Monien (34) untersuchte die Klasse RPP mit simultanen Zeit- und Platzschranken, bei der bereits eine Form von Parametrisierung zu sehen ist. Dabei versucht er, die gemeinsame Zeit- und Platzkomplexität eines Algorithmus durch Funktionen in Abhängigkeit von der Eingabelänge und eines Parameters zu beschränken. Diese Untersuchungen könnten zu einer dritten Hierarchie erweitert werden, die sich zwischen den oben betrachteten Zeit- und Platzhierarchien anordnet, die Zeit- und Platzkomplexität miteinander verbindet und damit die Komplexitätstheorie um eine neue Dimension erweitert.

## LITERATURVERZEICHNIS


[1] Alber, J. ; Bodlaender, H. L. ; Fernau, H. ; Kloks, T. ; Niedermeier, R.: Fixed Parameter Algorithms for Dominating Set and Related Problems on Planar Graphs. In: *Algorithmica* 33 (2002), Nr. 4, S. 461–493

[2] Arnborg, S. ; Corneil, D. G. ; Proskurowski, A.: Complexity of Finding Embeddings in a $k$-Tree. In: *SIAM Journal on Algebraic and Discrete Methods* 8 (1987), Nr. 2, S. 277–284

[3] Arnborg, S. ; Lagergren, J. ; Seese, D.: Easy Problems for Tree-Decomposable Graphs. In: *Journal of Algorithms* 12 (1991), April, S. 308–340

[4] Bellman, R. E.: *Dynamic Programming*. Dover Publications, 2003

[5] Bodlaender, H. L.: A Tourist Guide through Treewidth. In: *Acta Cybernetica* 11 (1993), Nr. 1-2, S. 1–23

[6] Bodlaender, H. L.: A linear time algorithm for finding tree-decompositions of small treewidth. In: *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, S. 226–234

[7] Borosh, I. ; Treybig, L. B.: Bounds on positive integral solutions of linear Diophantine equations. In: *Proceedings of the American Mathematical Society* 55 (1976), Nr. 2, S. 299–304

[8] Briggs, P. ; Cooper, K. D. ; Torczon, L.: Improvements to Graph Coloring Register Allocation. In: *ACM Transactions on Programming Languages Systems* 16 (1994), May, S. 428–455

[9] Błazewicz, J. ; Ecker, K. H. ; Schmidt, G. ; Węglarz, J.: *Scheduling in computer and manufacturing systems*. Second, Revised Edition. Springer, 1994

[10] Chaitin, G. J. ; Auslander, M. A. ; Chandra, A. K. ; Cocke, J. ; Hopkins, M. E. ; Markstein, P. W.: Register allocation via coloring. In: *Computer Languages* 6 (1981), Nr. 1, S. 47–57

[11] Chen, J. ; Kanj, I. ; Xia, G.: Improved parameterized upper bounds for vertex cover. In: *Mathematical Foundations of Computer Science 2006* (2006), S. 238–249

[12] Coffman Jr., E.G. ; Garey, M. R. ; Johnson, D. S.: Approximation algorithms for bin packing: a survey. In: *Approximation algorithms for NP-hard problems*. 1997, S. 46–93

[13] Cook, S. A.: The complexity of theorem-proving procedures. In: *STOC '71 Proceedings of the third annual ACM symposium on Theory of computing*, 1971, S. 151–158

[14] Courcelle, B.: Graph rewriting: an algebraic and logic approach. (1990), S. 193–242

[15] Diestel, R.: *Graphentheorie*. Springer, 1996

[16] Downey, R. G. ; Fellows, M. R.: *Parameterized complexity*. Bd. 5. Springer New York, 1999

[17] Elberfeld, M. ; Jakoby, A. ; Tantau, T.: Logspace Versions of the Theorems of Bodlaender and Courcelle. In: *Annual IEEE Symposium on Foundations of Computer Science* Bd. 0, 2010, S. 143–152

[18] Flum, J. ; Grohe, M.: *Parameterized complexity theory*. Springer, 2006

[19] Garey, M. R. ; Johnson, D. S.: *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1990

[20] Gill, P. E. ; Murray, W. ; Wright, M. H.: Numerical linear algebra and optimization. (1991)

[21] Harary, F.: *Graph Theory*. Addison-Wesley, 1969

[22] Harary, F. ; Ross, I. C.: A procedure for clique detection using the group matrix. In: *Sociometry* 20 (1957), Nr. 3, S. 205–215

[23] Hemaspaandra, L. A.: SIGACT News Complexity Theory Column 36. In: *SIGACT News* 33 (2002), S. 34–47

[24] Immerman, Neil: *Descriptive Complexity*. Springer, 1999

[25] Impagliazzo, R. ; Naor, M.: Efficient Cryptographic Schemes Provably as Secure as Subset Sum. In: *Journal of Cryptology* 9 (1996), S. 199–216

[26] Jakoby, A. ; Liskiewicz, M. ; Reischuk, R.: Space Efficient Algorithms for Directed Series-Parallel Graphs. In: *Journal of Algorithms* 60 (2006), Nr. 2, S. 85–114

[27] Jakoby, A. ; Tantau, T.: Logspace Algorithms for Computing Shortest and Longest Paths in Series-Parallel Graphs. In: *FSTTCS 2007: Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science* Bd. 4855, Springer, 2007, S. 216–227

[28] Jansen, K. ; Kratsch, S. ; Marx, D. ; Schlotter, I.: Bin Packing with Fixed Number of Bins Revisited. In: *Algorithm Theory – SWAT 2010: Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory* Bd. 6139, Springer, 2010, S. 260–272

[29] Jia, L. ; Rajaraman, R. ; Suel, T.: An efficient distributed algorithm for constructing small dominating sets. In: *Distributed Computing* 15 (2002), S. 193–205

[30] Karp, R. M.: Reducibility among Combinatorial Problems. In: *Proceedings of a Symposium on the Complexity of Computer Computations*, Plenum Publishing Corporation, 1972, S. 85–103

[31] Lokshtanov, D. ; Nederlof, J.: Saving space by algebraization. In: *STOC 2010: Proceedings of the 42nd ACM symposium on Theory of computing*, 2010, S. 321–330

[32] Luce, R. ; Perry, A.: A method of matrix analysis of group structure. In: *Psychometrika* 14 (1949), S. 95–116

[33] Martello, S. ; Toth, P.: *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, 1990

[34] Monien, B.: On a subclass of pseudopolynomial problems. In: *Symposium on Mathematical Foundations of Computer Science* Bd. 88. Springer, 1980, S. 414–425

[35] Odlyzko, A. M.: The rise and fall of knapsack cryptosystems. In: *Cryptology and computational number theory* 42 (1990), S. 75–88

[36] Papadimitriou, C. H.: On the Complexity of Integer Programming. In: *Journal of the Association for Computing Machinery* 28 (1981), Nr. 4, S. 765–768

[37] Papadimitriou, C. H.: *Computational Complexity*. John Wiley and Sons Ltd., 2003

[38] Reischuk, K. R.: *Komplexitätstheorie Band 1: Grundlagen*. Teubner, 1999

[39] Robertson, N. ; Seymour, P. D.: Graph minors. III. Planar tree-width. In: *Journal of Combinatorial Theory, Series B* 36 (1984), Nr. 1, S. 49–64

[40] Schneider, J. ; Wattenhofer, R.: A New Technique For Distributed Symmetry Breaking. In: *Proceeding of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, 2010, S. 257–266

[41] Takamizawa, K. ; Nishizeki, T. ; Saito, N.: Linear-Time Computability of Combinatorial Problems on Series-Parallel Graphs. In: *Journal of the Association for Computing Machinery* 29 (1982), Nr. 3, S. 623–641

[42] Vazirani, V. V.: *Approximation algorithms*. Springer, 2001

[43] Wu, J. ; Li, H.: A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks. In: *Telecommunication Systems* 18 (2001), S. 13–36