# 5

# Komplexitätstheorie

Wann ist ein Problem schwer? Warum ist ein Problem schwerer als ein anderes? Mit diesen Fragen befasst sich die Komplexitätstheorie. In Abschnitt 3.4 wurden ausgewählte Graphenprobleme vorgestellt, wie zum Beispiel das Färbbarkeitsproblem für Graphen, und es wurde erwähnt, dass alle diese Probleme „schwer“ sind. Was darunter zu verstehen ist, wird Inhalt dieses Kapitels sein.

Abschnitt 5.1 gibt eine kurze Einführung in die klassische Komplexitätstheorie. Abschnitt 5.2 stellt einen der möglichen Ansätze vor, mit der klassischen Berechnungshärte von Problemen umzugehen: die parametrisierte Komplexitätstheorie.

## 5.1 Klassische Komplexitätstheorie

### 5.1.1 Deterministische Zeit- und Platzklassen

Ein Problem, das durch einen Algorithmus in Polynomialzeit gelöst werden kann, ist effizient lösbar (siehe Abschnitt 2.4.3). Die Menge aller solcher Probleme fasst man in der Komplexitätsklasse P (deterministische Polynomialzeit) zusammen. In Beispiel 2.2 wurde ein Algorithmus vorgestellt, der das Zweifärbbarkeitsproblem in quadratischer Zeit (siehe Übung 2.3) bzw. sogar in Linearzeit (siehe Übung 2.9) löst und somit zeigt, dass dieses Problem zu P gehört.

deterministische Polynomialzeit P

Komplexitätsklassen sind Mengen von Problemen, deren Lösung bezüglich eines Komplexitätsmaßes (wie zum Beispiel Zeit- oder Platzbedarf) und eines Algorithmentyps (wie zum Beispiel Determinismus oder Nichtdeterminismus) etwa denselben algorithmischen Aufwand erfordert, eine vorgegebene Komplexitätsschranke (siehe Abschnitt 2.2) also nicht überschreitet. Bei der Klasse P ist diese Schranke ein beliebiges Polynom in der Eingabegröße. Allgemeiner definiert man die deterministischen Komplexitätsklassen für die Maße Zeit und Platz folgendermaßen, wobei wir daran erinnern, dass $T_A^{\mathrm{WC}}$ die Worst-case-Zeitkomplexität eines Algorithmus $A$ (siehe Definition 2.6) und $S_A^{\mathrm{WC}}$ die Worst-case-Platzkomplexität von $A$ (siehe Definition 2.7) bezeichnet.

F. Gurski et al., *Exakte Algorithmen für schwere Graphenprobleme*, eXamen.press,
DOI 10.1007/978-3-642-04500-4_5, 

**Definition 5.1 (deterministische Zeit- und Platzklassen).** *Für (berechenbare) Komplexitätsfunktionen $t$ und $s$, die von $\mathbb{N}$ in $\mathbb{N}$ abbilden, ist die* Klasse $\mathrm{DTIME}(t)$ der deterministisch in der Zeit $t$ lösbaren Probleme *durch*

$\mathrm{DTIME}(t)$

$$\mathrm{DTIME}(t) = \left\{ L \;\middle|\; \begin{array}{l} \textit{es gibt einen Algorithmus } A, \textit{ der } L \textit{ löst,} \\ \textit{und für alle } n \textit{ gilt } T_A^{WC}(n) \leq t(n) \end{array} \right\}$$

$\mathrm{DSPACE}(s)$ *und die* Klasse $\mathrm{DSPACE}(s)$ der deterministisch mit Platzaufwand $s$ lösbaren Probleme *durch*

$$\mathrm{DSPACE}(s) = \left\{ L \;\middle|\; \begin{array}{l} \textit{es gibt einen Algorithmus } A, \textit{ der } L \textit{ löst,} \\ \textit{und für alle } n \textit{ gilt } S_A^{WC}(n) \leq s(n) \end{array} \right\}$$

*definiert.*

Gemäß Definition 5.1 kann die Klasse P formal durch

$$\mathrm{P} = \bigcup_{k \geq 0} \mathrm{DTIME}(n^k)$$

definiert werden. Tabelle 5.1 listet neben P einige weitere deterministische Komplexitätsklassen auf, die wichtig genug sind, einen eigenen Namen zu verdienen. Oft lässt man dabei auch Familien von Komplexitätsfunktionen zu (wie etwa die Familie aller Polynome für die Klasse P), nicht nur einzelne solche Funktionen. Das ist sinnvoll, da sich (unter bestimmten Voraussetzungen) nach Satz 5.2 unten Platz linear komprimieren bzw. Zeit linear beschleunigen lässt. Beispielsweise sind die Klassen $\mathrm{DSPACE}(n)$ und $\mathrm{DSPACE}(17 \cdot n + 3)$ identisch und ebenso die Klassen $\mathrm{DTIME}(n^2)$ und $\mathrm{DTIME}(224 \cdot n^2 + 13)$.

**Tabelle 5.1.** Einige deterministische Komplexitätsklassen

| Zeitklassen | Platzklassen |
|---|---|
| $\mathrm{REALTIME} = \mathrm{DTIME}(n)$ | $\mathrm{L} = \mathrm{DSPACE}(\log)$ |
| $\mathrm{LINTIME} = \mathrm{DTIME}(\mathcal{O}(n))$ | $\mathrm{LINSPACE} = \mathrm{DSPACE}(\mathcal{O}(n))$ |
| $\mathrm{P} = \bigcup_{k \geq 0} \mathrm{DTIME}(n^k)$ | $\mathrm{PSPACE} = \bigcup_{k \geq 0} \mathrm{DSPACE}(n^k)$ |
| $\mathrm{E} = \mathrm{DTIME}(2^{\mathcal{O}(n)})$ | $\mathrm{ESPACE} = \mathrm{DSPACE}(2^{\mathcal{O}(n)})$ |
| $\mathrm{EXP} = \bigcup_{k \geq 0} \mathrm{DTIME}(2^{n^k})$ | $\mathrm{EXPSPACE} = \bigcup_{k \geq 0} \mathrm{DSPACE}(2^{n^k})$ |

Nicht alle Zeitklassen haben dabei ein Pendant hinsichtlich des Platzes, und nicht zu allen Platzklassen gibt es eine entsprechende Zeitklasse. Beispielsweise ist, wie bereits erwähnt, der Begriff der deterministischen logarithmischen Zeit nicht sinnvoll, da ein derart beschränkter Algorithmus nicht einmal jedes Symbol der Eingabe lesen könnte. Ebenso hat die Klasse REALTIME keine Entsprechung bei den Platzklassen, weil nach der zweiten Aussage von Satz 5.2 die Gleichheit

$$\mathrm{DSPACE}(n) = \mathrm{DSPACE}(\mathcal{O}(n)) = \mathrm{LINSPACE} \tag{5.1}$$

gilt, die (5.1) entsprechende Gleichheit bei den Zeitklassen jedoch nach einem Resultat von Rosenberg [Ros67] beweisbar nicht gilt:

$$\text{REALTIME} \neq \text{LINTIME}. \tag{5.2}$$

Es stellt sich die Frage, wie sich die Klassen in Tabelle 5.1 zueinander verhalten, unter welchen Bedingungen sie echt voneinander separiert sind und wie sich die Komplexitätsmaße Zeit und Platz ineinander umrechnen lassen. Wir geben die entsprechenden Resultate ohne Beweis an und verweisen interessierte Leser auf die einschlägige Literatur (wie z. B. [Pap94, Rot08]). Wenn wir im Folgenden von einer Komplexitätsfunktion sprechen, nehmen wir stillschweigend immer an, dass es sich um eine berechenbare Funktion handelt, die sich durch einen Algorithmus mit einem der jeweiligen Aussage entsprechenden Aufwand (bezüglich Zeit oder Platz) erzeugen lässt.[22]

Der folgende Satz sagt, dass sich ein Algorithmus $A$, der für Eingaben der Größe $n$ den Platz $s(n)$ benötigt, stets durch einen Algorithmus $A'$ simulieren lässt, der für eine von $A$ abhängige Konstante $c < 1$ mit dem Platz $c \cdot s(n)$ auskommt. Man spricht dann von linearer Platzkompression. Beispielsweise gilt $\text{DSPACE}(2 \cdot n^2) = \text{DSPACE}(n^2)$ nach der zweiten Aussage von Satz 5.2. Die entsprechende Aussage gilt auch für das Komplexitätsmaß Zeit, allerdings nur für solche Komplexitätsfunktionen $t$, die asymptotisch stärker als die Identität wachsen (siehe (5.2)). In diesem Fall spricht man von linearer Beschleunigung. Beide Aussagen lassen sich durch geeignete Codierungstricks beweisen.

**Satz 5.2 (lineare Beschleunigung und Kompression).**

1. *Für jede Komplexitätsfunktion $t$ mit $n \in o(t(n))$ gilt:*

$$\text{DTIME}(t) = \text{DTIME}(\mathcal{O}(t)).$$

2. *Für jede Komplexitätsfunktion $s$ gilt:*

$$\text{DSPACE}(s) = \text{DSPACE}(\mathcal{O}(s)).$$

**ohne Beweis**

Es stellt sich die Frage, um wie viel die Komplexitätsressource Zeit oder Platz wachsen muss, um eine echt größere Komplexitätsklasse zu erhalten. Diese Frage beantworten die Hierarchiesätze für Zeit und Platz, siehe Satz 5.3. Für den Platz ist die Antwort einfach: Nach der zweiten Aussage von Satz 5.2 gilt $\text{DSPACE}(s_2) \subseteq \text{DSPACE}(s_1)$ für alle Platzfunktionen $s_1$ und $s_2$ mit $s_2 \in \mathcal{O}(s_1)$. Satz 5.3 sagt, dass $\text{DSPACE}(s_2)$ jedoch nicht mehr in $\text{DSPACE}(s_1)$ enthalten ist, sobald $s_2 \notin \mathcal{O}(s_1)$

[22] Man sagt dann, die Komplexitätsfunktion ist „zeitkonstruierbar“ bzw. „platzkonstruierbar“. Dies sind vernünftige und notwendige Forderungen, ohne die die formalen Beweise nicht funktionieren würden. Da wir uns jedoch nicht in technischen Details verlieren wollen, verweisen wir wieder auf [Rot08] und merken hier lediglich an, dass alle „normalen“ oder „üblichen“ Komplexitätsfunktionen diese Konstruierbarkeitsbedingungen erfüllen.

gilt.[23] Für die Komplexitätsressource Zeit ist die Antwort etwas komplizierter, weil hier aus technischen Gründen (auf die wir hier nicht eingehen wollen) noch ein logarithmischer Faktor hinzukommt.

**Satz 5.3 (Hierarchiesätze).**

1. *Sind $t_1$ und $t_2$ Komplexitätsfunktionen, sodass $t_2 \notin \mathcal{O}(t_1)$ und $t_2(n) \geq n$ für alle $n$ gilt, so gilt:*
$$\mathrm{DTIME}(t_2 \log t_2) \not\subseteq \mathrm{DTIME}(t_1).$$
2. *Sind $s_1$ und $s_2$ Komplexitätsfunktionen mit $s_2 \notin \mathcal{O}(s_1)$, so gilt:*
$$\mathrm{DSPACE}(s_2) \not\subseteq \mathrm{DSPACE}(s_1).$$

**ohne Beweis**

Aus den Hierarchiesätzen für Zeit und Platz (sowie aus der Ungleichheit (5.2)) erhalten wir sofort die folgenden echt aufsteigenden Inklusionsketten – so genannte Hierarchien – von Komplexitätsklassen. Dabei bedeutet $\mathscr{C} \subset \mathscr{D}$ für Klassen $\mathscr{C}$ und $\mathscr{D}$, dass $\mathscr{C} \subseteq \mathscr{D}$ und $\mathscr{C} \neq \mathscr{D}$ gilt.

**Korollar 5.4.** *1.* $\mathrm{DTIME}(n) \subset \mathrm{DTIME}(n^2) \subset \mathrm{DTIME}(n^3) \subset \cdots$.
*2.* $\mathrm{DTIME}(2^n) \subset \mathrm{DTIME}(2^{2\cdot n}) \subset \mathrm{DTIME}(2^{3\cdot n}) \subset \cdots$.
*3.* $\mathrm{REALTIME} \subset \mathrm{LINTIME} \subset \mathrm{P} \subset \mathrm{E} \subset \mathrm{EXP}$.
*4.* $\mathrm{DSPACE}(n) \subset \mathrm{DSPACE}(n^2) \subset \mathrm{DSPACE}(n^3) \subset \cdots$.
*5.* $\mathrm{DSPACE}(2^n) \subset \mathrm{DSPACE}(2^{2\cdot n}) \subset \mathrm{DSPACE}(2^{3\cdot n}) \subset \cdots$.
*6.* $\mathrm{L} \subset \mathrm{LINSPACE} \subset \mathrm{PSPACE} \subset \mathrm{ESPACE} \subset \mathrm{EXPSPACE}$.

Kommen wir nun zur Frage, wie sich die Komplexitätsmaße Zeit und Platz ineinander umrechnen lassen. Natürlich kann man mit jedem Rechenschritt höchstens einen neuen Speicherplatz benutzen. Folglich ist die Zeitschranke eines Algorithmus $A$ zugleich immer auch eine Platzschranke für $A$, und die Lösung eines Problems durch $A$ erfordert somit höchstens so viel Platz wie Zeit. Umgekehrt kann man zeigen, dass die Zeitschranke eines Algorithmus höchstens exponentiell in seinem Platzbedarf ist. Das heißt, lässt sich ein Problem mit Platzbedarf $s \in \Omega(\log)$ lösen, so kann es für eine geeignete Konstante $c > 0$ in der Zeit $2^{c \cdot s}$ gelöst werden.

**Satz 5.5.** *1.* $\mathrm{DTIME}(t) \subseteq \mathrm{DSPACE}(t)$.
*2. Für $s \in \Omega(\log)$ gilt* $\mathrm{DSPACE}(s) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(s)})$. **ohne Beweis**

[23] Man beachte, dass $s_2 \notin \mathcal{O}(s_1)$ *nicht* äquivalent zu $s_2 \in o(s_1)$ ist, denn $s_2 \notin \mathcal{O}(s_1)$ sagt lediglich, dass $s_2$ unendlich oft stärker als $s_1$ wächst, wohingegen $s_2 \in o(s_1)$ bedeutet, dass $s_2$ fast überall (also für alle bis auf endlich viele Argumente) stärker als $s_1$ wächst (vgl. Übung 2.14).

### 5.1.2 Naiver Exponentialzeit-Algorithmus für Dreifärbbarkeit

Komplexitätsklassen und ihre Hierarchien bilden einen formalen Rahmen, um Ordnung in die algorithmische Komplexität der (mathematisch formulierbaren) Probleme dieser Welt zu bringen; sie sind sozusagen die Hülsen, die wir nun mit Inhalt füllen wollen. Die Elemente einer Komplexitätsklasse sind Probleme – nicht etwa Algorithmen, auch wenn die Zugehörigkeit eines Problems zu einer Komplexitätsklasse natürlich durch einen geeigneten Algorithmus bezeugt wird. Sehen wir uns eines der Probleme, die uns bereits begegnet sind, in dieser Hinsicht genauer an, nämlich das $k$-Färbbarkeitsproblem für Graphen.

In Beispiel 2.2 wurde ein Algorithmus vorgestellt, der das Zweifärbbarkeitsproblem in quadratischer Zeit (siehe Übung 2.3) bzw. sogar in Linearzeit (siehe Übung 2.9) löst. Demnach ist dieses Problem in der Klasse P enthalten:

**Behauptung 5.6.** 2-FÄRBBARKEIT *ist in* P.

Nun wollen wir die (deterministische) Komplexität des Dreifärbbarkeitsproblems untersuchen. Wie in Abschnitt 3.4.2 beschrieben, sucht dieses Problem die Antwort auf die Frage, ob es möglich ist, die Knotenmenge eines gegebenen Graphen $G = (V,E)$ in höchstens drei Farbklassen aufzuteilen. Wir suchen also eine Partition von $V$ in höchstens drei unabhängige Mengen.

Der unerfahrene Graphfärber würde vielleicht versuchen, den naiven Algorithmus anzuwenden, der einfach alle Lösungsmöglichkeiten der Reihe nach durchprobiert. Er erzeugt also nacheinander alle möglichen Partitionen von $V$ in höchstens drei Mengen und testet, ob alle Elemente einer solchen Partition unabhängige Mengen des Graphen sind. Findet er eine solche Partition, so akzeptiert er die Eingabe; sind alle solchen Partitionen erfolglos getestet worden, so lehnt er die Eingabe ab.

**Beispiel 5.7 (naiver Exponentialzeit-Algorithmus für Dreifärbbarkeit).** Sehen wir uns einen Testlauf dieses Algorithmus an. Abbildung 5.1 zeigt den durch die vier Knoten Edgar, Max, Moritz und Paul induzierten Teilgraphen des Graphen aus Abb. 1.1.

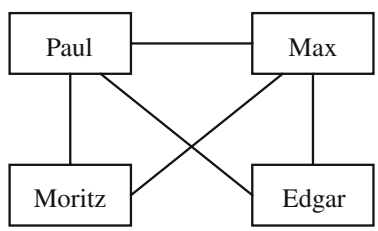



**Abb. 5.1.** Ein induzierter Teilgraph des Graphen aus Abb. 1.1

Zunächst testet der naive Algorithmus, ob der Graph mit nur einer Farbe färbbar ist. Es gibt natürlich nur eine solche Partition, deren einziges Element die ganze Knotenmenge des Graphen ist:

$$\{\{\text{Edgar}, \text{Max}, \text{Moritz}, \text{Paul}\}\}.$$

Offenbar ist diese Menge nicht unabhängig (siehe Abb. 5.1).

Dann testet der naive Algorithmus die Partitionen in zwei nicht leere Teilmengen der Knotenmenge, also ob der Graph zweifärbbar ist. Es gibt sieben solche Partitionen:

$$\{\{\text{Edgar}\}, \{\text{Max}, \text{Moritz}, \text{Paul}\}\}; \quad \{\{\text{Edgar}, \text{Max}\}, \{\text{Moritz}, \text{Paul}\}\};$$
$$\{\{\text{Max}\}, \{\text{Edgar}, \text{Moritz}, \text{Paul}\}\}; \quad \{\{\text{Edgar}, \text{Paul}\}, \{\text{Max}, \text{Moritz}\}\};$$
$$\{\{\text{Moritz}\}, \{\text{Edgar}, \text{Max}, \text{Paul}\}\}; \quad \{\{\text{Edgar}, \text{Moritz}\}, \{\text{Max}, \text{Paul}\}\};$$
$$\{\{\text{Paul}\}, \{\text{Edgar}, \text{Max}, \text{Moritz}\}\}.$$

Wie Abb. 5.2 zeigt, besteht keine dieser Partitionen nur aus unabhängigen Mengen.

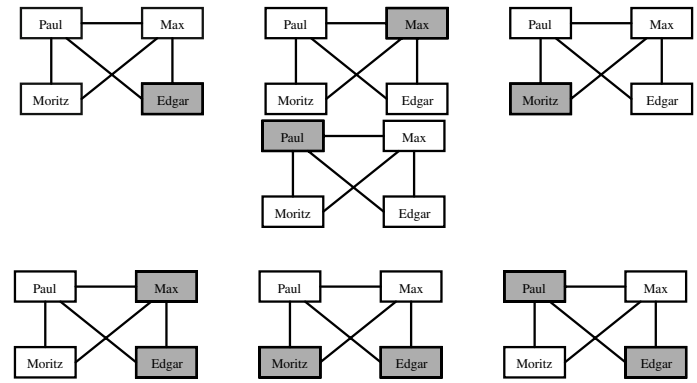



**Abb. 5.2.** Partitionen in zwei nicht leere Teilmengen

Schließlich testet der naive Algorithmus die Partitionen in drei nicht leere Teilmengen der Knotenmenge, also ob der Graph dreifärbbar ist. Es gibt sechs solche Partitionen:

$$\{\{\text{Edgar}, \text{Max}\}, \{\text{Moritz}\}, \{\text{Paul}\}\}; \quad \{\{\text{Edgar}, \text{Moritz}\}, \{\text{Max}\}, \{\text{Paul}\}\};$$
$$\{\{\text{Edgar}, \text{Paul}\}, \{\text{Max}\}, \{\text{Moritz}\}\}; \quad \{\{\text{Max}, \text{Moritz}\}, \{\text{Edgar}\}, \{\text{Paul}\}\};$$
$$\{\{\text{Max}, \text{Paul}\}, \{\text{Edgar}\}, \{\text{Moritz}\}\}; \quad \{\{\text{Moritz}, \text{Paul}\}, \{\text{Edgar}\}, \{\text{Max}\}\}.$$

Wie Abb. 5.3 zeigt, gibt es unter diesen nur eine Partition, deren sämtliche Elemente unabhängige Mengen des Graphen sind, nämlich $\{\{\text{Edgar}, \text{Moritz}\}, \{\text{Max}\}, \{\text{Paul}\}\}$ (die mittlere Partition in der oberen Reihe von Abb. 5.3).

Es ist klar, dass dieser Algorithmus korrekt ist: Existiert eine Lösung der gegebenen Probleminstanz, so findet er sie und akzeptiert; andernfalls lehnt er ab. Auch klar ist, dass die Laufzeit dieses Algorithmus von der Anzahl der möglichen Partitionen der Knotenmenge $V$, $|V| = n$, in höchstens drei disjunkte nicht leere Teilmengen abhängt.

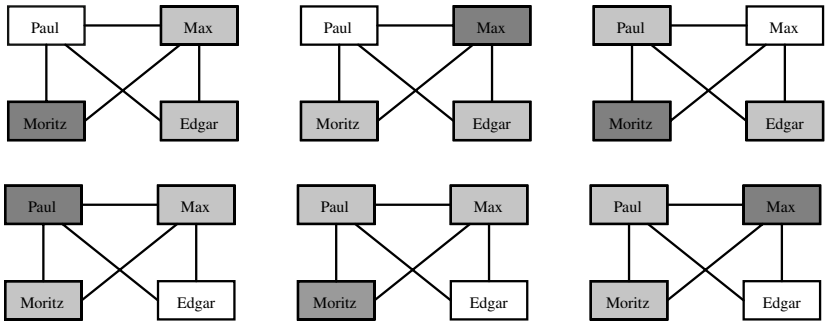



**Abb. 5.3.** Partitionen in drei nicht leere Teilmengen

Wie viele Partitionen einer $n$-elementigen Menge in $k \leq n$ nicht leere Teilmengen gibt es? Diese Frage wurde in der Kombinatorik gestellt und beantwortet, und die gesuchte Zahl ist dort so wichtig, dass sie einen eigenen Namen hat: die *Stirling-Zahl zweiter Art*, benannt nach dem Mathematiker James Stirling und bezeichnet mit $\left\{ {n \atop k} \right\}$ oder $S_{n,k}$. Man kann sich leicht die folgende Rekurrenz für diese Zahlen überlegen:

Stirling-Zahl zweiter Art $\left\{ {n \atop k} \right\}$

$$\left\{ {n \atop k} \right\} = \left\{ {n-1 \atop k-1} \right\} + k \cdot \left\{ {n-1 \atop k} \right\}. \tag{5.3}$$

**Übung 5.8.** Beweisen Sie, dass die Stirling-Zahlen zweiter Art die Rekurrenz (5.3) erfüllen. **Hinweis:** Eine Partition einer $n$-elementigen Menge in $k$ nicht leere Teilmengen enthält entweder die Menge $\{n\}$ als Element oder sie enthält sie nicht. Betrachten Sie diese beiden Fälle separat, um $\left\{ {n \atop k} \right\}$ zu bestimmen.

Für die Laufzeitanalyse des naiven Dreifärbbarkeitsalgorithmus interessiert uns natürlich die Zahl $\sum_{k=1}^{3} \left\{ {n \atop k} \right\}$, deren Terme wie folgt bestimmt werden können:

$$\left\{ {n \atop 1} \right\} = 1; \quad \left\{ {n \atop 2} \right\} = 2^{n-1} - 1; \quad \left\{ {n \atop 3} \right\} = \left(3^{n-1} - 2^{n-1}\right) - \frac{1}{2}\left(3^{n-1} - 1^{n-1}\right). \tag{5.4}$$

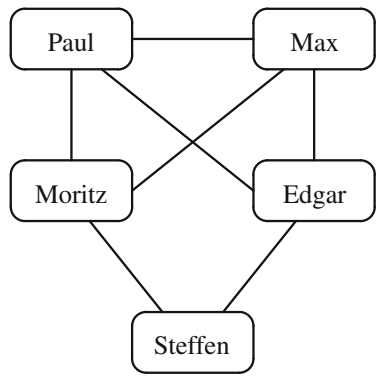
Dies ergibt sich aus der folgenden allgemeinen Formel für die explizite Darstellung der Stirling-Zahlen zweiter Art:

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{i=0}^{k} (-1)^{k-i} \binom{k}{i} i^n = \sum_{i=0}^{k} (-1)^{k-i} \frac{i^{n-1}}{(i-1)!(k-i)!}. \tag{5.5}$$

Dabei bezeichnet $k! = k(k-1)(k-2)\cdots 1$ die Fakultätsfunktion (also die Anzahl der Permutationen von $k$ Elementen) und $\binom{k}{i} = \frac{k!}{i!(k-1)!}$ den Binomialkoeffizienten (also die Anzahl der Möglichkeiten, $i$ Elemente aus einer $k$-elementigen Menge auszuwählen).

**Übung 5.9.** (a) Zeigen Sie, wie sich die Darstellungen für $\left\{ {n \atop 1} \right\}$, $\left\{ {n \atop 2} \right\}$ und $\left\{ {n \atop 3} \right\}$ in (5.4) aus der allgemeinen expliziten Formel (5.5) für die Stirling-Zahlen zweiter Art ergeben.

(b) Wie viele Partitionen in fünf nicht leere Teilmengen hat eine Menge mit $n \geq 5$ Elementen? Das heißt, bestimmen Sie $\left\{ {n \atop 5} \right\}$.
(c) Betrachten Sie den Graphen mit fünf Knoten aus Abb. 1.1:




Wie viele Partitionen in nicht leere Teilmengen der Knotenmenge dieses Graphen muss der naive Algorithmus für das Dreifärbbarkeitsproblem im schlimmsten Fall testen?
(d) Geben Sie alle diese Partitionen explizit an.

Wie wir in Beispiel 5.7 gesehen haben, ergeben sich für $n = 4$ in (5.4) tatsächlich die Werte $\left\{ {4 \atop 1} \right\} = 1$ (siehe Abb. 5.1), $\left\{ {4 \atop 2} \right\} = 2^{4-1} - 1 = 7$ (siehe Abb. 5.2) und

$$\left\{ {4 \atop 3} \right\} = \left(3^{4-1} - 2^{4-1}\right) - \frac{1}{2}\left(3^{4-1} - 1^{4-1}\right) = (27-8) - \frac{1}{2}(27-1) = 6$$

(siehe Abb. 5.3). Insgesamt mussten also für $n = 4$ in Beispiel 5.7 schlimmstenfalls $1+7+6 = 14$ Partitionen getestet werden.

Da man für jede Partition in Polynomialzeit testen kann, ob alle ihre Elemente unabhängige Mengen des Graphen sind, und da im schlimmsten Fall sämtliche Partitionen zu testen sind, ergibt sich die Laufzeit des naiven Dreifärbbarkeitsalgorithmus gemäß (5.4) im Wesentlichen durch

$$\begin{aligned}\left\{ {n \atop 1} \right\} + \left\{ {n \atop 2} \right\} + \left\{ {n \atop 3} \right\} &= 1 + 2^{n-1} - 1 + \left(3^{n-1} - 2^{n-1}\right) - \frac{1}{2}\left(3^{n-1} - 1^{n-1}\right) \\ &= 3^{n-1} - \frac{1}{2}\left(3^{n-1} - 1^{n-1}\right).\end{aligned}$$

Somit zeigt der naive Algorithmus, dass das Dreifärbbarkeitsproblem in der Komplexitätsklasse E liegt:

**Behauptung 5.10.** 3-FÄRBBARKEIT *ist in* E.

**Übung 5.11.** (a) Zeigen Sie, dass $3^n \in 2^{\mathcal{O}(n)}$ gilt.

(b) Zeigen Sie, dass $k$-FÄRBBARKEIT für jedes $k > 3$ in E liegt.

Eine der Behauptung 5.10 entsprechende Aussage gilt für jede der in Abschnitt 3.4 definierten Graphenprobleme: Sie alle sind durch den naiven Algorithmus in deterministischer Exponentialzeit lösbar. Können diese Probleme auch effizienter gelöst werden? Möglicherweise kann man für diese Probleme ja einen raffinierteren Algorithmus als den naiven finden. Schließlich würde der naive Algorithmus auch für die Lösung des Zweifärbbarkeitsproblems exponentielle Zeit brauchen, denn nach (5.4) ist auch die Anzahl $\left\{ {n \atop 2} \right\} = 2^{n-1} - 1$ der Partitionen in zwei nicht leere Teilmengen exponentiell in der Knotenzahl des Graphen. Doch das Zweifärbbarkeitsproblem hat, wie wir wissen, einen simplen Polynomialzeit-Algorithmus (siehe Beispiel 2.2, die Übungen 2.3 und 2.9 sowie Behauptung 5.6). Für das Dreifärbbarkeitsproblem ist es jedoch bis heute nicht gelungen, einen effizienten Algorithmus zu finden.

Der folgende Abschnitt gibt Indizien dafür an, weshalb das Dreifärbbarkeitsproblem und andere Probleme aus Abschnitt 3.4 vermutlich nicht in Polynomialzeit lösbar sind.

### 5.1.3 Nichtdeterminismus, Reduktionen und NP-Vollständigkeit

Um plausibel zu machen, weshalb es für bestimmte Probleme (wie die in Abschnitt 3.4 definierten) vermutlich keine Polynomialzeit-Algorithmen gibt, befassen wir uns nun mit Nichtdeterminismus und insbesondere der Komplexitätsklasse NP, definieren Reduktionen zwischen verschiedenen Problemen und erklären darauf aufbauend den Begriff der NP-Vollständigkeit. Man könnte analog zu Definition 5.1 auch die nichtdeterministischen Komplexitätsmaße – NTIME und NSPACE – formal einführen. Weil dieses Buch jedoch vorwiegend deterministische Algorithmen zum Inhalt hat und kein Buch über Komplexitätstheorie ist, verzichten wir darauf und führen den Begriff des Nichtdeterminismus stattdessen informal ein.

Ein nichtdeterministischer Algorithmus verfügt über eine besondere Gabe: Er kann Lösungen einer gegebenen Probleminstanz *raten*. Die Macht des Ratens ist schon seit den Gebrüdern Grimm bekannt:

„Heißt du Kaspar, Melchior, Balthasar?“,

und natürlich müssen geratene Lösungen – in einer deterministischen Phase – erst noch verifiziert werden:

„So heiß′ ich nicht!“.

Denn raten kann man vieles:

„Heißest du vielleicht Rippenbiest oder Hammelswade oder Schnuerbein?“,

doch nicht jede geratene Lösung ist auch eine korrekte Lösung der Probleminstanz:

„So heiß′ ich nicht!“.

Anders als im Märchen werden in der nichtdeterministischen Ratephase eines Algorithmus alle potentiellen Lösungen der Probleminstanz *parallel* – gleichzeitig also – geraten und anschließend parallel auf Korrektheit getestet. Somit kann man eine nichtdeterministische Berechnung als einen Baum darstellen, dessen Wurzel die Eingabe repräsentiert und dessen Wege von der Wurzel zu den Blättern dem Raten der Lösungskandidaten entsprechen. An den Blättern des Berechnungsbaumes wird die Korrektheit der jeweiligen Lösung getestet: „Heißest du Kunz?“ – „Nein.“ – „Heißest du Heinz?“ – „Nein.“ – „Heißt du etwa Rumpelstilzchen?“ Fußnote 24 gibt die Antwort. Abbildung 5.4 veranschaulicht diesen Vorgang.

Rumpelstilzchen

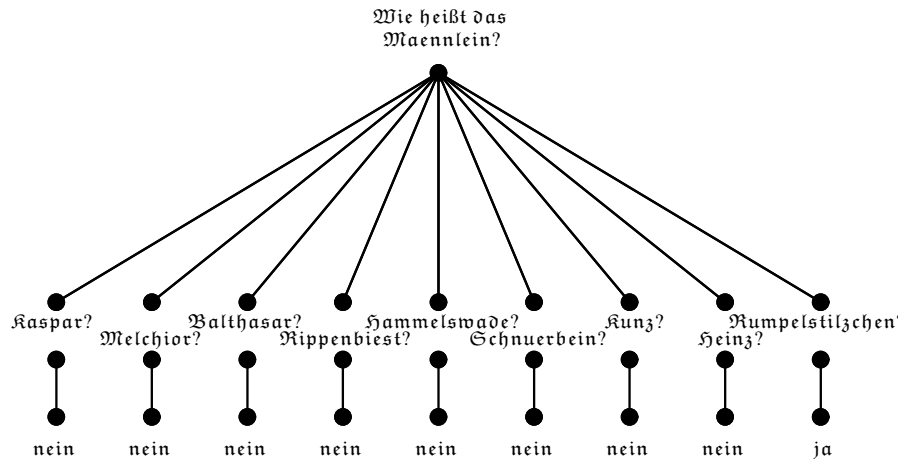



**Abb. 5.4.** Ein nichtdeterministischer Rateprozess mit deterministischer Verifikation

Bei einer nichtdeterministischen Berechnung ist zum Akzeptieren der Eingabe lediglich ein akzeptierender Weg im Berechnungsbaum erforderlich,[24] egal, wie viele der anderen Berechnungswege zur Ablehnung führen oder akzeptieren. Natürlich ist eine Eingabe auch bei mehreren akzeptierenden Berechnungen zu akzeptieren. Abgelehnt wird die Eingabe nur, wenn sämtliche Wege im Berechnungsbaum ablehnen.

nichtdeterministische Polynomialzeit NP

Die Komplexitätsklasse NP (nichtdeterministische Polynomialzeit) enthält genau die Probleme, die sich durch einen nichtdeterministischen Algorithmus in Polynomialzeit lösen lassen. Bei einem NP-Problem ist also sowohl die nichtdeterministische Ratephase als auch die deterministische Verifikationsphase polynomialzeit-

[24] Das ist die Stelle, wenn im Märchen das kleine Männlein schreit: „Das hat dir der Teufel gesagt, das hat dir der Teufel gesagt!“, seinen rechten Fuß vor Zorn tief in die Erde stößt, bis an den Leib hineinfährt, in seiner Wut seinen linken Fuß mit beiden Händen packt und sich selbst mitten entzwei reißt.

Ein nichtdeterministischer Algorithmus tut dies weitaus weniger drastisch, aber im Prinzip kann man sich sein Akzeptierungsverhalten ungefähr so vorstellen.

beschränkt. Alle in Abschnitt 3.4 definierten Graphenprobleme und viele Probleme der Aussagenlogik (siehe Kapitel 4) gehören zu NP. Das Raten von Lösungen erfolgt dabei in einer systematischen Art und Weise, sodass sämtliche möglichen Lösungskandidaten erfasst werden.

Zeuge, Zertifikat

Korrekte Lösungen der „Ja"-Instanzen eines NP-Problems nennt man auch *Zeugen* oder *Zertifikate* – sie „bezeugen" oder „zertifizieren", dass eine gegebene Instanz des Problems zur Menge der „Ja"-Instanzen dieses Problems gehört.[25] Da die Ratephase polynomiell beschränkt ist, ist die Länge eines Zertifikats immer polynomiell in der Eingabegröße beschränkt, und es lässt sich somit in Polynomialzeit erzeugen. Da die Verifikationsphase ebenfalls polynomiell beschränkt ist, kann man für jeden Lösungskandidaten einer Probleminstanz in Polynomialzeit bestimmen, ob er ein Zertifikat für diese ist. Demnach kann man die Probleme in NP folgendermaßen charakterisieren.

**Satz 5.12.** *Eine Menge $A$ gehört genau dann zu* NP*, wenn es eine Menge $B$ in* P *und ein Polynom $p$ gibt, sodass für alle Eingaben $x$ gilt:*

$$x \in A \Longleftrightarrow (\exists z : |z| \leq p(|x|))\,[(x,z) \in B].$$

**ohne Beweis**

**Übung 5.13.** Beweisen Sie Satz 5.12.

Betrachten wir nun konkret das Erfüllbarkeitsproblem der Aussagenlogik, SAT, das in Kapitel 4 definiert wurde.

**Behauptung 5.14.** SAT *ist in* NP.

**Beweis.** Um zu zeigen, dass SAT in NP liegt, geht ein nichtdeterministischer Polynomialzeit-Algorithmus (kurz: NP-Algorithmus) wie folgt vor. Bei Eingabe einer booleschen Formel $\varphi$ mit $n$ Variablen rät er eine Wahrheitswert-Belegung der Variablen von $\varphi$ und akzeptiert genau dann, wenn eine geratene Belegung die Formel erfüllt. Der dieser Berechnung entsprechende Baum besteht aus einem Binärbaum der Tiefe $n$ (denn für jede der $n$ Variablen wird eine Belegung mit 1 (wahr) oder 0 (falsch) geraten), an dessen $2^n$ Wege sich jeweils eine deterministische Berechnung anschließt, die polynomiell in $n$ ist (denn eine Formel kann für eine gegebene Belegung in Polynomialzeit ausgewertet werden). ❑

**Beispiel 5.15.** Abbildung 5.5 zeigt den Berechnungsbaum des NP-Algorithmus aus dem Beweis von Behauptung 5.14 konkret für die Eingabeformel

$$\varphi(x,y,z) = (x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z).$$

Die mit „A" markierten grauen Blätter dieses Baums stellen akzeptierende Berechnungen dar, die anderen ablehnende Berechnungen. Da es (mindestens) ein akzeptierendes Blatt gibt, ist die Eingabeformel $\varphi$ also erfüllbar. Die Belegungen $(0,0,0)$, $(1,0,0)$ und $(1,0,1)$ sind die Zertifikate dafür.

[25] Es ist üblich, Entscheidungsprobleme nicht nur durch ein „Gegeben"- und „Frage"-Feld wie in Abschnitt 3.4 anzugeben, sondern auch als Mengen der „Ja"-Instanzen darzustellen.

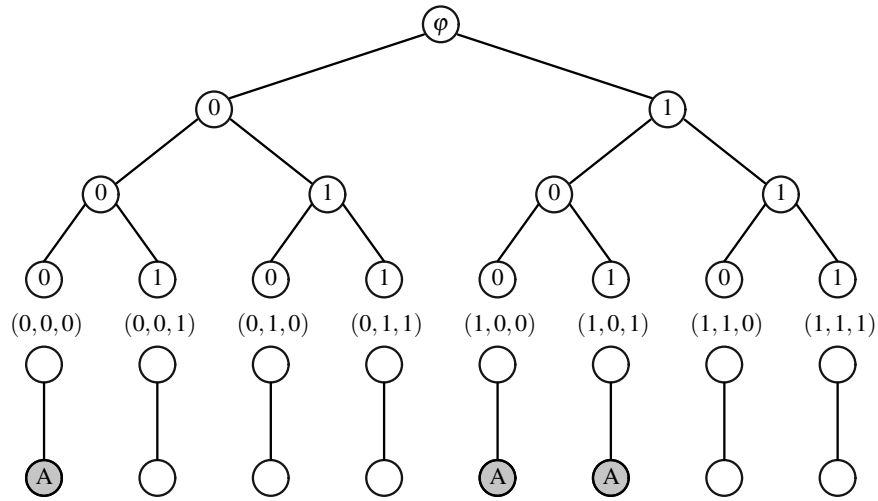



**Abb. 5.5.** Ein nichtdeterministischer Berechnungsbaum für die Formel $\varphi$ in Beipiel 5.15

Alternativ zum Beweis von Behauptung 5.14, der einen konkreten NP-Algorithmus für SAT beschreibt, kann man Satz 5.12 anwenden, um zu zeigen, dass SAT in NP liegt. Nach Definition ist eine boolesche Formel $\varphi$ in $n$ Variablen genau dann in SAT, wenn es eine erfüllende Belegung $z$ für $\varphi$ gibt, also ein Wort $z \in \{0,1\}$, $|z| = n$, mit $(\varphi, z) \in B$, wobei die Menge

$$B = \{(\varphi, z) \mid z \text{ ist eine erfüllende Belegung für die boolesche Formel } \varphi\}$$

offenbar in Polynomialzeit entscheidbar ist.

Wie oben erwähnt, gehören alle in Abschnitt 3.4 definierten Graphenprobleme ebenfalls zu NP. Wir zeigen dies nun konkret für das Dreifärbbarkeitsproblem. Den Beweis von Lemma 5.16 (das später im Beweis von Satz 5.26 benötigt wird) führen wir exemplarisch in Beispiel 5.17.

**Lemma 5.16.** 3-FÄRBBARKEIT *ist in* NP.

**Beispiel 5.17 (NP-Algorithmus für Dreifärbbarkeit).** Wir betrachten wieder den Graphen mit den vier Knoten Edgar, Max, Moritz und Paul aus Abb. 5.1. Der in Beispiel 5.7 beschriebene Exponentialzeit-Algorithmus für Dreifärbbarkeit listete deterministisch sämtliche Partitionen der Knotenmenge dieses Graphen in nicht leere Teilmengen auf (siehe Abb. 5.1, Abb. 5.2 und Abb. 5.3) und testete für jede Partition, ob alle ihre Elemente unabhängige Mengen des Graphen sind. Der NP-Algorithmus für Dreifärbbarkeit, den wir nun beschreiben wollen, geht im Grunde genauso vor, nur dass er die Partitionen parallel erzeugt und testet, nicht sequenziell. Für den gegebenen Graphen rät er nichtdeterministisch eine Partition der Knotenmenge in nicht leere Teilmengen und verifiziert deterministisch, ob sie eine legale Dreifärbung beschreibt. Abbildung 5.6 zeigt den resultierenden Berechnungsbaum. Das einzi-

ge Zertifikat, das die Dreifärbbarkeit des gegebenen Graphen belegt, ist die Partition $\{\{\text{Edgar}, \text{Moritz}\}, \{\text{Max}\}, \{\text{Paul}\}\}$, die zu dem mit „A“ markierten grauen Blatt führt, das als einziges akzeptiert; alle anderen Blätter lehnen ab.

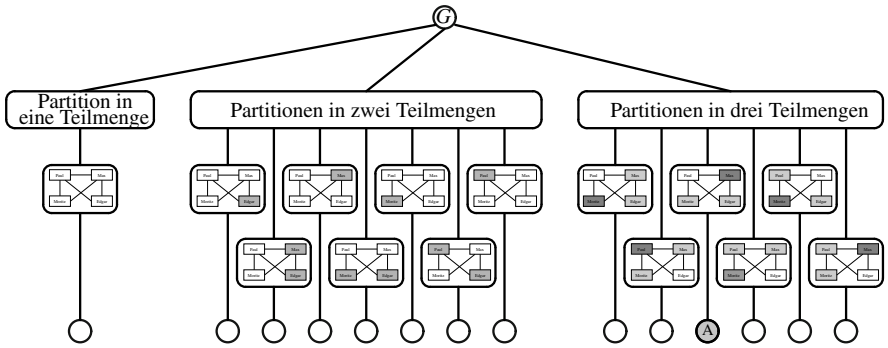



**Abb. 5.6.** NP-Algorithmus für Dreifärbbarkeit

Weshalb liegt 3-FÄRBBARKEIT vermutlich nicht in der Klasse P? Dieses Problem liegt in E und in NP, die beide P enthalten, aber die Zugehörigkeit eines Problems zu einer Klasse verbietet natürlich nicht seine Zugehörigkeit zu einer kleineren Teilklasse. Auch die leere Menge – das einfachste Problem überhaupt – liegt in E und in NP (und dennoch auch in viel kleineren Komplexitätsklassen).

Bevor wir uns der im vorherigen Absatz gestellten Frage zuwenden, befassen wir uns noch kurz mit weiteren nichtdeterministischen Komplexitätsklassen. Die zu den Sätzen über lineare Beschleunigung und Kompression (siehe Satz 5.2) und den Hierarchiesätzen (siehe Satz 5.3) analogen Aussagen gelten auch für die nichtdeterministischen Maße für Zeit und Platz, und neben NP gibt es noch viele weitere nichtdeterministische Komplexitätsklassen. Als Beispiele greifen wir hier nur zwei nichtdeterministische Platzklassen heraus: NL (nichtdeterministischer logarithmischer Platz) und NPSPACE (nichtdeterministischer polynomieller Platz). Nach einem Resultat von Savitch [Sav70] ist jedoch bekannt, dass

$$\text{PSPACE} = \text{NPSPACE} \tag{5.6}$$

gilt. Die folgende Inklusionskette von Komplexitätsklassen zeigt, wie sich Determinismus und Nichtdeterminismus zueinander und bezüglich der Maße Zeit und Platz verhalten:

$$\text{L} \subseteq \text{NL} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE}. \tag{5.7}$$

Von keiner der Inklusionen in (5.7) weiß man, ob sie echt ist; lediglich die echte Inklusion $\text{NL} \subset \text{PSPACE}$ folgt aus dem o. g. Satz von Savitch (bzw. aus dessen Folgerung (5.6)) und dem Hierarchiesatz für nichtdeterministische Platzklassen.

Wie man an den Beispielen 5.7 und 5.17 sieht, können NP-Algorithmen deterministisch in Exponentialzeit simuliert werden. Das heißt, der naive deterministische

Algorithmus aus Beispiel 5.7 führt gewissermaßen eine Tiefensuche durch den Berechnungsbaum des NP-Algorithmus für Dreifärbbarkeit (siehe Abb. 5.6) aus. Kann man beliebige NP-Algorithmen auch in deterministischer Polynomialzeit simulieren? Diese faszinierende Frage ist seit etwa vier Jahrzehnten ungelöst und stellt eines der bedeutendsten offenen Probleme der Informatik dar, das so genannte „P = NP?"-Problem. Es wird allgemein vermutet, dass P ≠ NP gilt, aber anders als bei der bekannten Ungleichheit P ≠ E, die aus Satz 5.3 folgt (siehe Korollar 5.4), steht ein Beweis der Ungleichheit P ≠ NP nach wie vor aus.

Für den Nachweis, dass ein Problem zu einer Komplexitätsklasse gehört, ist „lediglich"[26] ein geeigneter Algorithmus zu finden, der dem Typ der Klasse entspricht (z. B. deterministisch oder nichtdeterministisch ist) und mit dem der Klasse entsprechenden Zeit- oder Platzbedarf auskommt. So wurde etwa in den Beispielen 5.7 und 5.17 durch geeignete Algorithmen gezeigt, dass das Dreifärbbarkeitsproblem zu den Klassen E und NP gehört. Die Existenz eines geeigneten Algorithmus zeigt also eine *obere Komplexitätsschranke*.

Zum Nachweis, dass ein Problem *nicht* in einer bestimmten Komplexitätsklasse liegt, muss man hingegen *jeden* Algorithmus, der diese Klasse repräsentiert, schlagen und zeigen, dass nicht einer davon das betrachtete Problem lösen kann. Das ist oft noch weitaus mühsamer als der Nachweis einer oberen Schranke. Um gegen die Gesamtheit aller Algorithmen des jeweiligen Typs vorzugehen, die in einer bestimmten Zeit oder einem bestimmten Platz arbeiten, bedarf es anderer Techniken als beim Algorithmenentwurf, und diese Techniken stoßen schnell an ihre Grenzen. Dies erklärt zum Teil die Schwierigkeit, sich einer Lösung des „P = NP?"-Problems zu nähern.

Eine etwas bescheidenere Aufgabe besteht darin, zu zeigen, dass ein Problem zwar zu einer Komplexitätsklasse gehört, aber *mindestens* so schwer wie jedes andere Problem dieser Klasse ist. Man spricht dann von einer *unteren Schranke* und sagt, das Problem ist *hart* für die Klasse (bzw. sogar *vollständig* für die Klasse, sofern das Problem zur Klasse gehört). Diese Begriffe sind wichtig, um unser Ziel zu erreichen: Indizien dafür zu finden, dass das Dreifärbbarkeitsproblem und andere Graphenprobleme aus Abschnitt 3.4 vermutlich nicht in Polynomialzeit lösbar sind.

Um zwei gegebene Probleme hinsichtlich ihrer Komplexität zu vergleichen, führen wir nun den Begriff der Reduzierbarkeit ein.

**Definition 5.18.** *Seien $A$ und $B$ zwei Mengen, beide über demselben Alphabet $\Sigma$ codiert, d. h., die Elemente von $A$ und $B$ (bzw. die Instanzen der Probleme $A$ und $B$) sind Wörter mit Buchstaben aus $\Sigma$. Mit $\Sigma^*$ bezeichnen wir die Menge aller Wörter über $\Sigma$. Sei $\mathscr{C}$ eine Komplexitätsklasse.*

$\leq_m^p$-Reduzierbarkeit

1. *$A$ ist* (in Polynomialzeit many-one-)reduzierbar auf *$B$, bezeichnet mit $A \leq_m^p B$, falls es eine in Polynomialzeit berechenbare Funktion $f$ gibt, sodass für alle $x \in \Sigma^*$ gilt:*
$$x \in A \iff f(x) \in B.$$

$\mathscr{C}$-hart

2. *$B$ ist* (bezüglich der $\leq_m^p$-Reduzierbarkeit) $\mathscr{C}$-hart *(synonym: $\mathscr{C}$-schwer), falls $A \leq_m^p B$ für jede Menge $A$ in $\mathscr{C}$ gilt.*

[26] Auch das kann natürlich sehr schwierig sein.

*3. Ist $B$ in $\mathscr{C}$ und $\mathscr{C}$-hart (bezüglich $\leq_m^p$), so heißt $B$ $\mathscr{C}$-*vollständig (bezüglich $\leq_m^p$). $\mathscr{C}$-vollständig

Unter einer Reduktion eines Problems $A$ auf ein Problem $B$ kann man sich eine Transformation aller Instanzen[27] von $A$ in Instanzen von $B$ vorstellen, die leicht (nämlich in Polynomialzeit) ausgeführt werden kann. In gewissem Sinn „verkleidet“ diese Transformation die Instanzen von $A$ als Instanzen von $B$. Daraus folgt sofort, dass, wenn $A \leq_m^p B$ gilt und $B$ in Polynomialzeit gelöst werden kann, auch $A$ in P ist. Das bedeutet, dass die Klasse P bezüglich der $\leq_m^p$-Reduzierbarkeit abgeschlossen ist. Anders gesagt, obere Schranken vererben sich bezüglich der $\leq_m^p$-Reduzierbarkeit nach unten. Ebenso kann man sich leicht überlegen, dass sich untere Schranken bezüglich der $\leq_m^p$-Reduzierbarkeit nach oben vererben, d. h., wenn $A \leq_m^p B$ gilt und $A$ NP-hart ist, so ist auch $B$ NP-hart. Diese grundlegenden Eigenschaften der $\leq_m^p$-Reduzierbarkeit werden in den ersten beiden Aussagen von Satz 5.19 genannt. Die dritte Aussage ist ebenfalls leicht zu zeigen; sie sagt, dass es zur Lösung des „P = NP?“-Problems genügt, für lediglich ein NP-vollständiges Problem zu zeigen, dass es in P liegt.

**Satz 5.19.** *1. Gilt $A \leq_m^p B$ und ist $B$ in* P*, so ist auch $A \in$ P.*
*2. Gilt $A \leq_m^p B$ und ist $A$* NP*-hart, so ist auch $B$* NP*-hart.*
*3.* P = NP *gilt genau dann, wenn ein* NP*-vollständiges Problem in* P *liegt.*
**ohne Beweis**

**Übung 5.20.** Beweisen Sie Satz 5.19.

Das erste natürliche Problem, dessen NP-Vollständigkeit gezeigt werden konnte, ist das Erfüllbarkeitsproblem der Aussagenlogik, SAT. Dieses bahnbrechende Resultat, das auf Steven Cook [Coo71] zurückgeht, geben wir ohne Beweis im folgenden Satz an. Der Beweis beruht auf einer Konstruktion, die die Berechnung eines NP-Algorithmus in eine boolesche Formel codiert, sodass die Formel genau dann erfüllbar ist, wenn der Algorithmus akzeptiert.

**Satz 5.21 (Cook [Coo71]).** SAT *ist* NP*-vollständig.* **ohne Beweis**

Die NP-Vollständigkeit von SAT wurde mittels der zweiten Aussage von Satz 5.19 auf zahlreiche weitere Probleme vererbt (siehe Garey und Johnson [GJ79]). Oft ist es dabei nützlich, ausgehend von einer geeigneten Einschränkung von SAT auf das Problem zu reduzieren, dessen NP-Vollständigkeit gezeigt werden soll. Natürlich muss dafür diese eingeschränkte SAT-Variante ebenfalls NP-vollständig sein. In Kapitel 4 wurde beispielsweise das Problem 3-SAT definiert, das wie SAT NP-vollständig ist, wie die folgende Behauptung zeigt.

**Behauptung 5.22.** 3-SAT *ist* NP*-vollständig.*

[27] Sowohl der „Ja“- als auch der „Nein“-Instanzen von $A$ – und zu letzteren gehören auch alle Eingabewörter, die syntaktisch inkorrekt und daher unmittelbar abzulehnen sind.

**Beweis.** Für den Nachweis der NP-Vollständigkeit eines Problems sind gemäß Definition 5.18 sowohl eine obere Schranke (es liegt in NP) als auch eine untere Schranke (es ist NP-hart) zu zeigen. Dass 3-SAT in NP liegt, folgt unmittelbar aus $\mathrm{SAT} \in \mathrm{NP}$.

Die NP-Härte von 3-SAT ergibt sich aus der folgenden Überlegung. Jeder boolesche Ausdruck in konjunktiver Normalform kann wie folgt sehr einfach in einen äquivalenten booleschen Ausdruck in konjunktiver Normalform transformiert werden, in dem jede Klausel genau drei Literale enthält. Dazu werden neue Hilfsvariablen $h_{i,j}$ benötigt.

Eine Klausel $C_i = (\ell_{i,1})$ mit genau einem Literal $\ell_{i,1}$ wird durch eine Teilformel mit vier Klauseln ersetzt:

$$(\ell_{i,1} \vee h_{i,1} \vee h_{i,2}) \wedge (\ell_{i,1} \vee h_{i,1} \vee \neg h_{h,2}) \wedge (\ell_{i,1} \vee \neg h_{i,1} \vee h_{i,2}) \wedge (\ell_{i,1} \vee \neg h_{i,1} \vee \neg h_{i,2}).$$

Eine Klausel $C_i = (\ell_{i,1} \vee \ell_{i,2})$ mit genau zwei Literalen wird durch eine Teilformel mit zwei Klauseln ersetzt:

$$(\ell_{i,1} \vee \ell_{i,2} \vee h_{i,1}) \wedge (\ell_{i,1} \vee \ell_{i,2} \vee \neg h_{i,1}).$$

Eine Klausel $C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \cdots \vee \ell_{i,k})$, $k > 3$, mit mehr als drei Literalen wird durch eine Teilformel mit $k-2$ Klauseln ersetzt:

$$\begin{aligned}&(\ell_{i,1} \vee \ell_{i,2} \vee h_{i,1}) \wedge (\ell_{i,3} \vee \neg h_{i,1} \vee h_{i,2}) \wedge \cdots \\ &\cdots \wedge (\ell_{i,k-2} \vee \neg h_{i,k-4} \vee h_{i,k-3}) \wedge (\ell_{i,k-1} \vee \ell_{i,k} \vee \neg h_{i,k-3}).\end{aligned}$$

Die Variablen $h_{i,j}$ sind, wie gesagt, neue Hilfsvariablen, die in dem ursprünglichen Ausdruck nicht verwendet wurden. Der dadurch entstehende Ausdruck besitzt offensichtlich genau dann eine erfüllende Belegung, wenn es für den ursprünglichen Ausdruck $\varphi$ eine erfüllende Belegung gibt. Die Transformation der gegebenen SAT-Instanz in eine äquivalente 3-SAT-Instanz ist sehr effizient in linearer Zeit durchführbar und zeigt $\mathrm{SAT} \leq_{\mathrm{m}}^{\mathrm{p}} \text{3-SAT}$. Nach Satz 5.21 ist 3-SAT NP-hart und somit NP-vollständig. ❑

**Beispiel 5.23 (Reduktion von SAT auf 3-SAT).** Sei

$$\varphi = (x_1) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5)$$

ein boolescher Ausdruck mit der Variablenmenge $X(\varphi) = \{x_1, x_2, x_3, x_4, x_5\}$. Dann ist

$$\begin{aligned}\varphi' = {} & (x_1 \vee h_{1,1} \vee h_{1,2}) \wedge (x_1 \vee h_{1,1} \vee \neg h_{1,2}) \wedge \\ & (x_1 \vee \neg h_{1,1} \vee h_{1,2}) \wedge (x_1 \vee \neg h_{1,1} \vee \neg h_{1,2}) \wedge \\ & (\neg x_1 \vee x_2 \vee h_{2,1}) \wedge (\neg x_1 \vee x_2 \vee \neg h_{2,1}) \wedge \\ & (x_1 \vee \neg x_2 \vee h_{3,1}) \wedge (\neg x_3 \vee \neg h_{3,1} \vee h_{3,2}) \wedge (x_4 \vee \neg x_5 \vee \neg h_{3,2})\end{aligned}$$

ein zu $\varphi$ äquivalenter boolescher Ausdruck mit Variablenmenge

$$X(\varphi') = \{x_1, x_2, x_3, x_4, x_5, h_{1,1}, h_{1,2}, h_{2,1}, h_{3,1}, h_{3,2}\},$$

in dem jede Klausel genau drei Literale enthält.

Eine weitere Einschränkung von SAT ist das folgende Problem:

NOT-ALL-EQUAL-3-SAT
NAE-3-SAT

| NOT-ALL-EQUAL-3-SAT (NAE-3-SAT) | |
|---|---|
| *Gegeben:* | Eine boolesche Formel $\varphi$ in konjunktiver Normalform, sodass jede Klausel genau drei Literale hat. |
| *Frage:* | Gibt es eine erfüllende Belegung für $\varphi$, die in keiner Klausel sämtliche Literale wahr macht? |

Unsere Reduktion im Beweis von Satz 5.26 wird von dem eingeschränkten Erfüllbarkeitsproblem NAE-3-SAT ausgehen, das ebenfalls NP-vollständig ist (siehe Übung 5.25(a)).

**Lemma 5.24.** NAE-3-SAT *ist* NP*-vollständig.* **ohne Beweis**

**Übung 5.25.** (a) Beweisen Sie Lemma 5.24. **Hinweis:** [Sch78].
(b) Zeigen Sie, dass $k$-FÄRBBARKEIT für jedes $k > 3$ NP-vollständig ist.
(c) Zeigen Sie, dass die in Abschnitt 3.4.3 definierten Entscheidungsprobleme DOMINIERENDE MENGE und DOMATISCHE ZAHL NP-vollständig sind.

**Satz 5.26.** 3-FÄRBBARKEIT *ist* NP*-vollständig.*

**Beweis.** Dass 3-FÄRBBARKEIT in NP liegt, ist bereits aus Lemma 5.16 bekannt. Um die NP-Härte von 3-FÄRBBARKEIT zu zeigen, reduzieren wir das nach Lemma 5.24 NP-vollständige Problem NAE-3-SAT auf 3-FÄRBBARKEIT.

Sei eine boolesche Formel $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in konjunktiver Normalform gegeben, sodass jede Klausel $C_j$ von $\varphi$ genau drei Literale hat. Seien $x_1, x_2, \ldots, x_n$ die Variablen von $\varphi$. Für $j \in \{1, 2, \ldots, m\}$ habe die $j$-te Klausel die Form $C_j = (\ell_{j1} \vee \ell_{j2} \vee \ell_{j3})$, wobei $\ell_{jk}$, $1 \leq k \leq 3$, ein Literal über den Variablen $x_1, x_2, \ldots, x_n$ ist, d. h., $\ell_{jk} = x_i$ oder $\ell_{jk} = \neg x_i$ für ein $i$, $1 \leq i \leq n$.

Konstruiere einen Graphen $G_\varphi = (V, E)$ wie folgt. Die Knotenmenge von $G_\varphi$ ist

$$V = \{u\} \cup \bigcup_{i=1}^{n} \{v_i, v_i'\} \cup \bigcup_{j=1}^{m} \{c_{j1}, c_{j2}, c_{j3}\},$$

d. h., für jede Variable $x_i$ werden zwei Knoten, $v_i$ und $v_i'$, und für jede der drei Positionen $k \in \{1, 2, 3\}$ in einer jeden Klausel $C_j$ wird ein Knoten $c_{jk}$ eingeführt, und außerdem gibt es einen speziellen Knoten $u$. Die Kantenmenge $E$ von $G_\varphi$ ist so definiert:

1. Für jede Variable $x_i$, $1 \leq i \leq n$, gibt es drei Kanten: $\{v_i, v_i'\}$, $\{v_i, u\}$ und $\{v_i', u\}$. Den durch $\{u, v_i, v_i'\}$, $1 \leq i \leq n$, induzierten Teilgraphen von $G_\varphi$ nennen wir ein *Variablendreieck* und die Knoten $v_i$ bzw. $v_i'$ *Variablenknoten*.
2. Für jede Klausel $C_j$, $1 \leq j \leq m$, gibt es drei Kanten: $\{c_{j1}, c_{j2}\}$, $\{c_{j1}, c_{j3}\}$ und $\{c_{j2}, c_{j3}\}$. Den durch $\{c_{j1}, c_{j2}, c_{j3}\}$, $1 \leq j \leq m$, induzierten Teilgraphen von $G_\varphi$ nennen wir ein *Klauseldreieck*.

3. Die so definierten Variablen- und Klauseldreiecke von $G_\varphi$ werden nun durch Kanten verbunden: Für $i \in \{1,2,\ldots,n\}$, $j \in \{1,2,\ldots,m\}$ und $k \in \{1,2,3\}$ gibt es genau dann eine Kante zwischen $v_i$ und $c_{jk}$, wenn $\ell_{jk} = x_i$ ein positives Literal ist, und es gibt genau dann eine Kante zwischen $v'_i$ und $c_{jk}$, wenn $\ell_{jk} = \neg x_i$ ein negatives Literal ist.

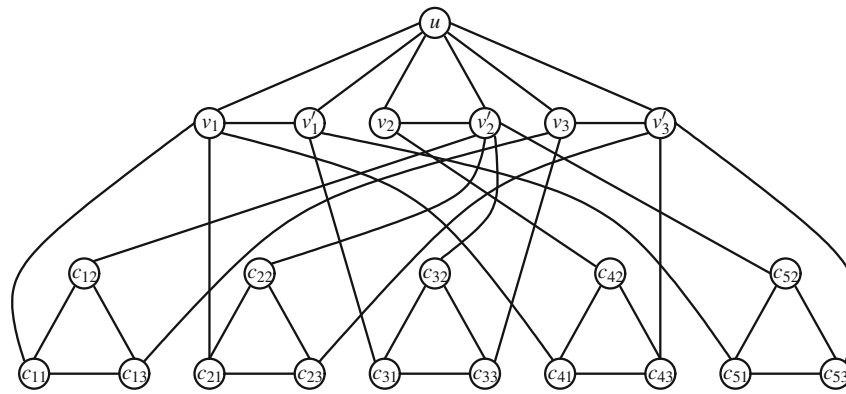



**Abb. 5.7.** Reduktion von NAE-3-SAT auf 3-FÄRBBARKEIT

Abbildung 5.7 zeigt den Graphen $G_\varphi$ für die Formel

$$\varphi(x_1,x_2,x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3),$$

die wir schon aus Beispiel 5.15 (mit anderen Variablennamen) kennen.

Offensichtlich lässt sich $G_\varphi$ in Polynomialzeit aus $\varphi$ konstruieren. Gemäß Definition 5.18 ist die folgende Äquivalenz zu zeigen:

$$\varphi \in \text{NAE-3-SAT} \iff G_\varphi \in \text{3-FÄRBBARKEIT}. \tag{5.8}$$

Angenommen, $\varphi \in$ NAE-3-SAT. Dann gibt es eine Belegung $z$, die $\varphi$ erfüllt, also in jeder Klausel mindestens ein Literal wahr macht, die aber gleichzeitig auch in jeder Klausel mindestens ein Literal falsch macht. Mit Hilfe von $z$ können wir den Graphen $G_\varphi$ folgendermaßen färben. Belegt $z$ eine Variable $x_i$ mit dem Wahrheitswert wahr, so färben wir $v_i$ mit der Farbe 1 und $v'_i$ mit der Farbe 2; macht $z$ dagegen $\neg x_i$ wahr (also $x_i$ falsch), so färben wir $v_i$ mit der Farbe 2 und $v'_i$ mit der Farbe 1. Wird nun der Knoten $u$ mit der Farbe 3 gefärbt, so sind alle Variablendreiecke legal dreigefärbt. Da $z$ in jeder Klausel ein Literal wahr macht und eines falsch, ist in jedem Klauseldreieck von $G_\varphi$ ein Eckknoten zu einem mit 1 gefärbten Variablenknoten adjazent (und wird deshalb selbst mit 2 gefärbt) und ein anderer Eckknoten zu einem mit 2 gefärbten Variablenknoten adjazent (und wird deshalb selbst mit 1 gefärbt). Für den dritten Eckknoten eines jeden Klauseldreiecks ist somit noch die Farbe 3 verfügbar. Also ist $G_\varphi$ dreifärbbar.

Sei umgekehrt angenommen, dass $G_\varphi$ dreifärbbar ist, und sei $\psi : V \to \{1,2,3\}$ eine legale Dreifärbung von $G_\varphi$. Wir konstruieren mit Hilfe von $\psi$ eine Belegung für $\varphi$. Ohne Beschränkung der Allgemeinheit dürfen wir annehmen, dass $\psi(u) = 3$ gilt. Dann müssen alle Variablenknoten die Farbe entweder 1 oder 2 haben, wobei zwei benachbarte Variablenknoten (also $v_i$ und $v_i'$, $1 \leq i \leq n$) verschieden gefärbt sind. Dies liefert eine Wahrheitswert-Belegung $z$ der Variablen von $\varphi$: Für $i \in \{1,2,\ldots,n\}$ ist $x_i$ unter $z$ genau dann wahr, wenn $v_i$ die Farbe 1 hat. Weil $\psi$ eine legale Dreifärbung von $G_\varphi$ ist, müssen insbesondere die drei Eckknoten in jedem Klauseldreieck verschieden gefärbt sein. Daraus folgt, dass $z$ in jeder Klausel von $\varphi$ ein Literal wahr und ein Literal falsch macht. Somit ist $\varphi$ in NAE-3-SAT.

Folglich gilt die Äquivalenz (5.8), und NAE-3-SAT $\leq_{\mathrm{m}}^{\mathrm{p}}$ 3-FÄRBBARKEIT ist gezeigt. Nach Lemma 5.24 und der zweiten Aussage von Satz 5.19 ist 3-FÄRBBARKEIT NP-hart. ❑

**Beispiel 5.27 (Reduktion von NAE-3-SAT auf 3-Färbbarkeit).** Abbildung 5.7 zeigt den Graphen $G_\varphi$, der gemäß der Konstruktion im Beweis von Satz 5.26 aus der booleschen Formel

$$\varphi(x_1,x_2,x_3) = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

resultiert. Diese Formel ist *nicht* in NAE-3-SAT, denn wie wir in Beispiel 5.15 gesehen haben, sind $(0,0,0)$, $(1,0,0)$ und $(1,0,1)$ ihre einzigen erfüllenden Belegungen, und keine davon hat die „Not-all-equal"-Eigenschaft: Die Belegung $(0,0,0)$ macht alle Literale in der letzten Klausel $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$ wahr; die Belegung $(1,0,0)$ macht alle Literale in der zweiten Klausel $(x_1 \vee \neg x_2 \vee \neg x_3)$ wahr; und $(1,0,1)$ macht alle Literale in der ersten Klausel $(x_1 \vee \neg x_2 \vee x_3)$ wahr. Dementsprechend ist der Graph $G_\varphi$ nicht dreifärbbar. Würden wir zum Beispiel versuchen, $G_\varphi$ gemäß der Argumentation im Beweis von (5.8) ausgehend von der erfüllenden Belegung $(1,0,1)$ zu färben, so ergäbe sich das Problem, dass alle drei Eckknoten der ersten Klausel zu den mit der Farbe 1 gefärbten Variablenknoten $v_1$, $v_2'$ und $v_3$ adjazent sind, weshalb sie sich in einer legalen Dreifärbung nur der Farben 2 und 3 bedienen dürften. Das ist aber unmöglich, weil je zwei dieser Eckknoten auch miteinander verbunden sind. Abbildung 5.8 zeigt diese illegale Dreifärbung, in der die benachbarten Knoten $c_{11}$ und $c_{13}$ gleich gefärbt sind, wobei die Farbe 1 weiß, die Farbe 2 hellgrau und die Farbe 3 dunkelgrau dargestellt ist. Jede andere Dreifärbung von $G_\varphi$ ist ebenfalls illegal. Dies zeigt, dass die Implikation $\varphi \notin$ NAE-3-SAT $\Longrightarrow G_\varphi \notin$ 3-FÄRBBARKEIT (die Kontraposition der Implikation $G_\varphi \in$ 3-FÄRBBARKEIT $\Longrightarrow \varphi \in$ NAE-3-SAT aus Äquivalenz (5.8)) für die Beispielformel $\varphi$ gilt.

Wenn wir jedoch die Formel $\varphi$ leicht abändern, indem wir ihre erste Klausel streichen, so erhalten wir die neue Formel

$$\varphi'(x_1,x_2,x_3) = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3),$$

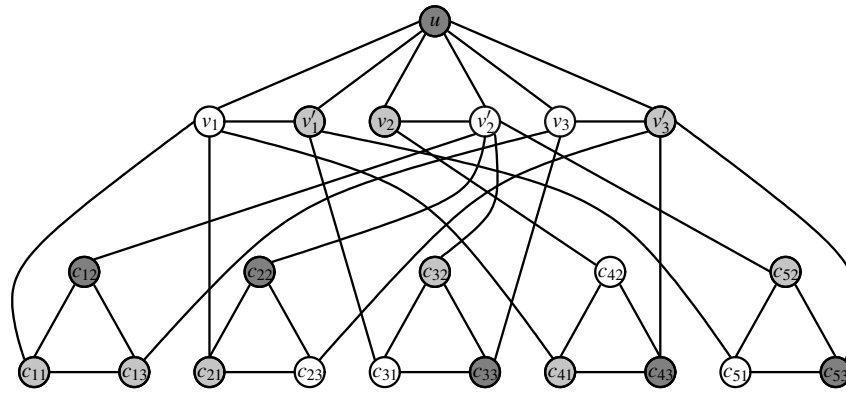



**Abb. 5.8.** Graph $G_\varphi$ ist wegen $\varphi \notin$ NAE-3-SAT nicht dreifärbbar.

die in NAE-3-SAT ist. Tatsächlich ist nun $(1,0,1)$ eine Belegung, die $\varphi'$ erfüllt und zusätzlich in allen Klauseln mindestens ein Literal falsch macht. Dementsprechend ist der Graph $G_{\varphi'}$, der sich aus $\varphi'$ gemäß der Konstruktion im Beweis von Satz 5.26 ergibt, dreifärbbar, und Abb. 5.9 zeigt eine legale Dreifärbung von $G_{\varphi'}$. Dies entspricht für die Beispielformel $\varphi'$ der Implikation $\varphi' \in$ NAE-3-SAT $\Longrightarrow$ $G_{\varphi'} \in$ 3-FÄRBBARKEIT aus Äquivalenz (5.8).

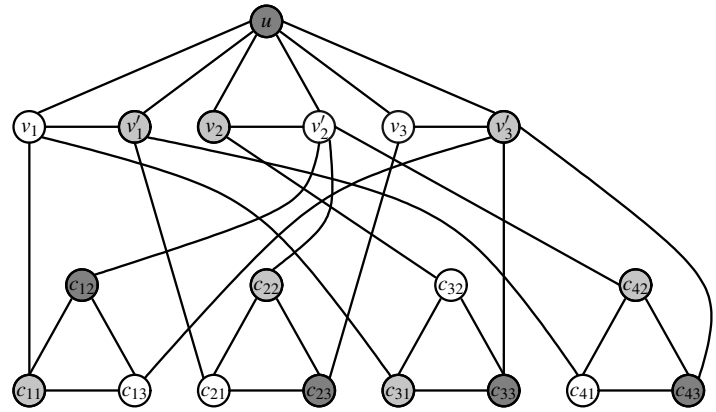



**Abb. 5.9.** Graph $G_{\varphi'}$ ist wegen $\varphi' \in$ NAE-3-SAT dreifärbbar.

Wie das Dreifärbbarkeitsproblem sind auch alle anderen der in Abschnitt 3.4 definierten Probleme NP-vollständig. Nun können wir die gesuchten Indizien dafür benennen, dass vermutlich keines dieser Probleme einen Polynomialzeit-Algorithmus besitzt. Wäre nämlich auch nur ein einziges dieser Probleme in Polynomialzeit lösbar, so würde nach der dritten Aussage von Satz 5.19 P = NP gelten, was jedoch

für sehr unwahrscheinlich gehalten wird. Es ist natürlich im Prinzip möglich, dass irgendwann die überraschende Gleichheit P = NP bewiesen wird. Versuche gab es in großer Zahl, doch jeder ist bisher gescheitert, sobald der „Beweis“ genauer überprüft wurde.

Es gibt jedoch noch mehr Gründe für die allgemeine Annahme, dass $P \neq NP$ gilt,[28] als nur die Tatsache, dass bisher jeder Beweisversuch für P = NP – und insbesondere jeder Versuch des Entwurfs eines Polynomialzeit-Algorithmus für die Probleme aus Abschnitt 3.4 – misslungen ist. Zum Beispiel hätte die Gleichheit P = NP unerwartete Auswirkungen auf die Struktur der Komplexitätsklassen und -hierarchien. Eine Reihe von Hierarchien – wie die so genannte Polynomialzeit-Hierarchie (siehe Abschnitt 5.1.4) oder die boolesche Hierarchie über NP (siehe z. B. [Rot08]) – würden kollabieren, wenn P = NP gelten würde. Wäre etwa das NP-vollständige Dreifärbbarkeitsproblem in P, so wären dann auch viele andere Probleme in P, die man für noch weitaus schwerer hält als das Dreifärbbarkeitsproblem. Die Intuition für $P \neq NP$ beruht gewissermaßen auf dem Glauben, dass sich das Grundprinzip der Diversität auch auf die Natur der algorithmischen Komplexität von Problemen erstreckt. Eine Welt, in der fast alle natürlichen Probleme einen Polynomialzeit-Algorithmus hätten, wäre zweifelsohne effizienter, aber sie wäre auch eine viel langweiligere Welt.

### 5.1.4 Die Polynomialzeit-Hierarchie

Es gibt eine Reihe von Komplexitätshierarchien über NP (siehe z. B. [Rot08]); die wichtigste unter diesen ist die von Meyer und Stockmeyer [MS72, Sto77] eingeführte Polynomialzeit-Hierarchie. Um diese Hierarchie zu definieren, benötigen wir zunächst einige Begriffe.

**Definition 5.28.** *Seien $\mathscr{C}$ und $\mathscr{D}$ zwei Komplexitätsklassen.*

1. *Die* Klasse der Komplemente von Problemen in $\mathscr{C}$ *ist definiert als* — co$\mathscr{C}$

$$\text{co}\mathscr{C} = \{\overline{C} \mid C \in \mathscr{C}\}.$$

2. *$\mathscr{C}^{\mathscr{D}}$ ist die Klasse der Probleme, die von einem Algorithmus im Sinne von $\mathscr{C}$ mit einer Orakelmenge $D \in \mathscr{D}$ akzeptiert werden können.*[29] *In einer solchen Berechnung ist es dem Algorithmus erlaubt, Fragen an seine Orakelmenge zu stellen, und er erhält die Antworten jeweils in einem Schritt, egal wie schwer die Orakelmenge zu bestimmen ist. Man kann sich eine solche Berechnung so vorstellen, dass für jede Frage eine Unterprozedur für D zu Einheitskosten ausgeführt wird; demgemäß spricht man von einer* Berechnung relativ zu *D.* — $\mathscr{C}^{\mathscr{D}}$; Orakelmenge; relative Berechnung

[28] Allerdings ist bisher auch kein Beweis der vermuteten Ungleichheit $P \neq NP$ geglückt.

[29] Formal wird dies durch den Begriff der *Orakel-Turingmaschine* definiert, siehe z. B. [Rot08]. Dies ist ein Algorithmenmodell, das zusätzlich mit einem Frageband ausgestattet ist, auf das der Algorithmus während seiner Berechnung Fragen schreiben kann. In einem speziellen Zustand, dem Fragezustand, erhält der Algorithmus die Antwort auf die aktuelle Frage (nämlich ob dieses Wort zur Orakelmenge gehört oder nicht) und setzt die Berechnung abhängig von dieser Antwort entweder im „Ja“- oder „Nein“-Zustand fort. — Orakel-Turingmaschine

coNP

Beispielsweise ist $\text{coNP} = \{\overline{A} \mid A \in \text{NP}\}$ die Klasse der Komplemente von NP-Problemen. Es ist nicht bekannt, ob $\text{NP} = \text{coNP}$ gilt; man vermutet allgemein, dass diese beiden Klassen verschieden sind. Mit anderen Worten, man vermutet, dass NP nicht unter Komplementbildung abgeschlossen ist. Das Tautologie-Problem ist ein Beispiel für ein Problem, das in coNP, aber vermutlich nicht in NP liegt (tatsächlich ist es coNP-vollständig, siehe Übung 5.29). Eine *Tautologie* ist ein boolescher Ausdruck, der *gültig* ist, d. h., der für jede Belegung seiner Variablen wahr ist. Beispielsweise ist $(x \wedge \neg x) \Rightarrow y$ eine Tautologie, da die Prämisse der Implikation, $(x \wedge \neg x)$, für jede Belegung der Variablen $x$ falsch, die Implikation also stets wahr ist.

Tautologie

TAUTOLOGY

| TAUTOLOGY | |
|---|---|
| *Gegeben:* | Ein boolescher Ausdruck $\varphi$. |
| *Frage:* | Ist $\varphi$ eine Tautologie? |

**Übung 5.29.** (a) Zeigen Sie, dass TAUTOLOGY coNP-vollständig ist.
**Hinweis:** Geben Sie eine Reduktion vom Komplement von SAT auf TAUTOLOGY an.

polynomialzeit-beschränkte Turing-Reduzierbarkeit

(b) Seien $A$ und $B$ zwei Probleme. Wir sagen, $A$ ist *in Polynomialzeit Turing-reduzierbar auf* $B$, falls $A \in \text{P}^B$ gilt (d. h., falls es einen deterministischen Polynomialzeit-Algorithmus gibt, der $A$ mit Fragen an das Orakel $B$ entscheidet).
Zeigen Sie, dass $A$ in Polynomialzeit Turing-reduzierbar auf $B$ ist, falls $A \leq_{\text{m}}^{\text{p}} B$ gilt. (Mit anderen Worten, die $\leq_{\text{m}}^{\text{p}}$-Reduzierbarkeit ist eine spezielle polynomialzeit-beschränkte Turing-Reduzierbarkeit.)

Die Klasse P jedoch ist – wie jede deterministische Klasse – unter Komplementbildung abgeschlossen, d. h., es gilt $\text{P} = \text{coP}$.

$\text{P}^{\text{NP}}$

Als ein konkretes Beispiel für $\mathscr{C}^{\mathscr{D}}$ betrachten wir $\text{P}^{\text{NP}}$, die Klasse der Probleme, die sich durch einen Polynomialzeit-Algorithmus entscheiden lassen, der während seiner Berechnung Fragen an eine NP-Menge stellen darf. Ein in $\text{P}^{\text{NP}}$ vollständiges Problem ist die folgende Variante von TSP:

UNIQUE OPTIMAL TRAVELING SALESPERSON PROBLEM

| UNIQUE OPTIMAL TRAVELING SALESPERSON PROBLEM | |
|---|---|
| *Gegeben:* | Ein vollständiger Graph $K_n = (V, E)$ und eine Funktion $c : E \to \mathbb{N}$. |
| *Frage:* | Gibt es in $K_n$ genau eine Rundreise kürzester Länge? |

UNIQUE OPTIMAL TRAVELING SALESPERSON PROBLEM

Die Klasse $\text{P}^{\text{NP}}$ enthält viele natürliche Varianten von Problemen in NP. Anders als bei den NP-Problemen, die in der Regel lediglich nach der Existenz einer Lösung fragen, geht es bei Problemen in $\text{P}^{\text{NP}}$ oft darum, zu entscheiden, ob eine optimale Lösung eine bestimmte Eigenschaft erfüllt, ob sie beispielsweise (wie bei UNIQUE OPTIMAL TRAVELING SALESPERSON PROBLEM) sogar eindeutig ist. Ein anderes

Beispiel für ein $P^{NP}$-vollständiges Problem ist die folgende Variante des Erfüllbarkeitsproblems: ODD-MAX-SAT

| ODD-MAX-SAT | |
|---|---|
| *Gegeben:* | Ein erfüllbarer boolescher Ausdruck $\varphi$. |
| *Frage:* | Ist die größte erfüllende Belegung der Variablen von $\varphi$ eine ungerade Zahl?[30] |

Häufig ist man auch an funktionalen Problemen (wie zum Beispiel dem in Übung 5.30 definierten Suchproblem) interessiert und betrachtetet dann die zur Klasse $P^{NP}$ von Entscheidungsproblemen analoge Funktionenklasse $FP^{NP}$.

**Übung 5.30.** Es sei $G = (V,E)$ ein planarer Graph mit der Knotenmenge $V = \{v_1, v_2, \ldots, v_n\}$. Appel and Haken [AH77a, AH77b] zeigten, dass jeder planare Graph vierfärbbar ist. Jede Vierfärbung $\psi_G : V \to \{1,2,3,4\}$ kann als ein Wort in $\{1,2,3,4\}^n$ aufgefasst werden, das durch $\psi_G = \psi_G(v_1)\psi_G(v_2)\cdots\psi_G(v_n)$ definiert ist. Zeigen Sie, dass das folgende Suchproblem in $FP^{NP}$ liegt: MIN-4-FÄRBBARKEIT FÜR PLANARE GRAPHEN

| MIN-4-FÄRBBARKEIT FÜR PLANARE GRAPHEN | |
|---|---|
| *Gegeben:* | Ein planarer Graph $G$. |
| *Ausgabe:* | Die lexikographisch kleinste legale Vierfärbung von $G$. |

**Hinweis:** Bestimmen Sie zunächst eine geeignete NP-Orakelmenge $A$ und führen Sie dann eine Präfixsuche auf $\{1,2,3,4\}^n$ durch, die die lexikographisch kleinste legale Vierfärbung von $G$ durch Fragen an $A$ ermittelt.

Die Klassen P, NP, coNP und $P^{NP}$ gehören zur Polynomialzeit-Hierarchie, die wir nun definieren.

**Definition 5.31 (Polynomialzeit-Hierarchie).** *Die* Stufen der Polynomialzeit-Hierarchie *sind induktiv wie folgt definiert:* $\Delta_i^p$ $\Sigma_i^p$ $\Pi_i^p$

$$\Delta_0^p = \Sigma_0^p = \Pi_0^p = \mathrm{P};$$
$$\Delta_{i+1}^p = \mathrm{P}^{\Sigma_i^p}, \quad \Sigma_{i+1}^p = \mathrm{NP}^{\Sigma_i^p} \quad \textit{und} \quad \Pi_{i+1}^p = \mathrm{co}\Sigma_{i+1}^p \quad \textit{für } i \geq 0.$$

*Die* Polynomialzeit-Hierarchie *ist die Vereinigung ihrer Stufen:* Polynomialzeit-Hierarchie PH

$$\mathrm{PH} = \bigcup_{i \geq 0} \Sigma_i^p.$$

[30] Dabei fasst man eine Belegung $T : X(\varphi) \to \{0,1\}$ als ein Wort der Länge $n = |X(\varphi)|$ über $\{0,1\}$ auf, wobei 0 für **false** und 1 für **true** steht, und betrachtet die lexikographische Ordnung auf $\{0,1\}^n$. Es wird also gefragt, ob das am wenigsten signifikante Bit der in dieser Ordnung größten erfüllenden Belegung von $\varphi$ eine 1 ist oder nicht.

Auf der nullten Stufe der Polynomialzeit-Hierarchie stimmen die Klassen $\Delta_0^p$, $\Sigma_0^p$ und $\Pi_0^p$ überein; sie sind alle gleich P. Auf der ersten Stufe unterscheiden sie sich jedoch vermutlich: Während $\Delta_1^p = \mathrm{P}^{\mathrm{P}} = \mathrm{P}$ ist, gilt $\Sigma_1^p = \mathrm{NP}^{\mathrm{P}} = \mathrm{NP}$ und $\Pi_1^p = \mathrm{coNP}^{\mathrm{P}} = \mathrm{coNP}$. Auf der zweiten Stufe, die aus den Klassen $\Delta_2^p = \mathrm{P}^{\mathrm{NP}}$, $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$ und $\Pi_2^p = \mathrm{coNP}^{\mathrm{NP}}$ besteht, und auf allen höheren Stufen sind die entsprechenden Klassen vermutlich ebenfalls verschieden. Ebenso sind alle Stufen der Hierarchie vermutlich verschieden, d. h., vermutlich gilt $\Delta_i^p \neq \Delta_{i+1}^p$, $\Sigma_i^p \neq \Sigma_{i+1}^p$ und $\Pi_i^p \neq \Pi_{i+1}^p$ für alle $i \geq 0$. „Vermutlich“ muss man hier sagen, weil es bis heute keinen Beweis irgendeiner dieser Separationen gibt. Gäbe es jedoch eine Gleichheit zwischen zwei aufeinander folgenden Stufen der Polynomialzeit-Hierarchie (oder auch nur die Gleichheit $\Sigma_i^p = \Pi_i^p$ für irgendein $i \geq 1$), so würde die gesamte Hierarchie auf diese Stufe kollabieren. Diese Eigenschaft und andere grundlegende Eigenschaften der Polynomialzeit-Hierarchie werden ohne Beweis im folgenden Satz zusammengefasst. Zunächst benötigen wir noch den Begriff des Abschlusses einer Komplexitätsklasse unter bestimmten Reduzierbarkeiten.

**Definition 5.32 (Abschluss unter Reduzierbarkeiten).**

Abschluss unter $\leq_{\mathrm{m}}^{\mathrm{p}}$-Reduktionen

1. *Eine Klasse $\mathscr{C}$ heißt* abgeschlossen unter $\leq_{\mathrm{m}}^{\mathrm{p}}$-Reduktionen*, falls aus $B \in \mathscr{C}$ und $A \leq_{\mathrm{m}}^{\mathrm{p}} B$ folgt, dass $A \in \mathscr{C}$ gilt.*

Abschluss unter Turing-Reduktionen

2. *Eine Klasse $\mathscr{C}$ heißt* abgeschlossen unter (polynomialzeit-beschränkten) Turing-Reduktionen*, falls* $\mathrm{P}^{\mathscr{C}} \subseteq \mathscr{C}$ *gilt.*

**Satz 5.33.** *1. Für jedes $i \geq 0$ gilt*

$$\Sigma_i^p \cup \Pi_i^p \subseteq \Delta_{i+1}^p \subseteq \Sigma_{i+1}^p \cap \Pi_{i+1}^p.$$

2. PH $\subseteq$ PSPACE*.*
3. *Jede der Klassen $\Delta_i^p$, $\Sigma_i^p$ und $\Pi_i^p$, $i \geq 0$, sowie* PH *ist abgeschlossen unter $\leq_{\mathrm{m}}^{\mathrm{p}}$-Reduktionen.*
4. *Die Stufen $\Delta_i^p$, $i \geq 0$, der Polynomialzeit-Hierarchie sind sogar unter polynomialzeit-beschränkten Turing-Reduktionen abgeschlossen (d. h., $\mathrm{P}^{\Delta_i^p} \subseteq \Delta_i^p$ für alle $i \geq 0$; siehe dazu auch Übung 5.29(b)).*
5. *Jede der Klassen $\Delta_i^p$, $\Sigma_i^p$ und $\Pi_i^p$, $i \geq 0$, hat $\leq_{\mathrm{m}}^{\mathrm{p}}$-vollständige Probleme.*
6. *Gäbe es ein $\leq_{\mathrm{m}}^{\mathrm{p}}$-vollständiges Problem für* PH*, dann würde diese Hierarchie auf eine endliche Stufe kollabieren.*
7. *Ist $\Sigma_i^p = \Pi_i^p$ für ein $i \geq 1$ oder ist $\Sigma_{i+1}^p = \Sigma_i^p$ für ein $i \geq 0$, dann kollabiert die* PH *auf ihre i-te Stufe:*

$$\Sigma_i^p = \Pi_i^p = \Sigma_{i+1}^p = \Pi_{i+1}^p = \cdots = \mathrm{PH}.$$

**ohne Beweis**

**Übung 5.34.** Beweisen Sie Satz 5.33.

Der Rateprozess einer NP-Berechnung entspricht einer (polynomiell längenbeschränkten) *existenziellen* Quantifizierung: Gegeben eine Probleminstanz $x$, gibt es

eine Lösung $y$ der Länge höchstens $p(|x|)$ für ein Polynom $p$? Der sich anschließende deterministische Verifikationsprozess einer NP-Berechnung prüft die Korrektheit der geratenen Lösung $y$. Diese Charakterisierung von NP wurde bereits in Satz 5.12 ausgedrückt. Der folgende Satz beschreibt eine analoge Charakterisierung der Klasse coNP durch (polynomiell längenbeschränkte) *universelle* Quantifizierungen.

**Satz 5.35.** *Eine Menge $A$ gehört genau dann zu* coNP*, wenn es eine Menge $B$ in* P *und ein Polynom $p$ gibt, sodass für alle Eingaben $x$ gilt:*

$$x \in A \iff (\forall z : |z| \leq p(|x|))\,[(x,z) \in B].$$

**ohne Beweis**

**Übung 5.36.** Beweisen Sie Satz 5.35.

Die Charakterisierungen von NP und coNP in den Sätzen 5.12 und 5.35 lassen sich folgendermaßen auf alle Stufen der Polynomialzeit-Hierarchie verallgemeinern.

**Satz 5.37.** *Sei $i \geq 1$.*

1. *Eine Menge $A$ gehört genau dann zu $\Sigma_i^p$, wenn es eine Menge $B$ in* P *und ein Polynom $p$ gibt, sodass für alle Eingaben $x$ gilt:*

$$\begin{aligned} x \in A \iff & (\exists z_1 : |z_1| \leq p(|x|))(\forall z_2 : |z_2| \leq p(|x|)) \cdots \\ & \cdots (\mathfrak{Q}_i z_i : |z_i| \leq p(|x|))\,[(x, z_1, z_2, \ldots, z_i) \in B], \end{aligned} \tag{5.9}$$

   *wobei $\mathfrak{Q}_i = \exists$ für ungerades $i$ und $\mathfrak{Q}_i = \forall$ für gerades $i$.*
2. *Eine Menge $A$ gehört genau dann zu $\Pi_i^p$, wenn es eine Menge $B$ in* P *und ein Polynom $p$ gibt, sodass für alle Eingaben $x$ gilt:*

$$\begin{aligned} x \in A \iff & (\forall z_1 : |z_1| \leq p(|x|))(\exists z_2 : |z_2| \leq p(|x|)) \cdots \\ & \cdots (\mathfrak{Q}_i z_i : |z_i| \leq p(|x|))\,[(x, z_1, z_2, \ldots, z_i) \in B], \end{aligned} \tag{5.10}$$

   *wobei $\mathfrak{Q}_i = \forall$ für ungerades $i$ und $\mathfrak{Q}_i = \exists$ für gerades $i$.*

**ohne Beweis**

**Übung 5.38.** Beweisen Sie Satz 5.37.

Die Äquivalenz (5.9) in Satz 5.37 hat eine schöne Interpretation im Zusammenhang mit strategischen Zwei-Personen-Spielen wie Schach. Man kann beispielsweise ein „Matt in drei Zügen“ folgendermaßen ausdrücken: Gegeben eine Spielsituation $x$, gibt es einen Zug $z_1$ von Weiß, sodass es für jeden Folgezug $z_2$ von Schwarz einen Zug $z_3$ von Weiß gibt, der Schwarz Matt setzt?

Nicht überraschend ist, dass die Quantorenstruktur der Klassen $\Sigma_i^p$ und $\Pi_i^p$ (und ebenso – für eine unbeschränkte Anzahl von alternierenden polynomiell längenbeschränkten $\exists$- und $\forall$-Quantoren – der Klasse PSPACE) vollständige Probleme aus dem Bereich der Logik (siehe Kapitel 4) in diesen Klassen liefert. So kann gemäß Satz 5.37 die Komplexität der Auswertung logischer Formeln erster Ordnung wie folgt den Komplexitätsklassen $\Sigma_i^p$, $\Pi_i^p$ und PSPACE zugeordnet werden. Wir erinnern dazu an die Definition des Entscheidungsproblems $\textsc{Dec}(\varphi)$ in Abschnitt 4.8 und an die Klassen $\Sigma_0^{\mathrm{FO}}$ und $\Pi_0^{\mathrm{FO}}$ von Formeln erster Ordnung in Definition 4.48.

**Behauptung 5.39.** *1. Für $\varphi \in \Sigma_i^{FO}$ ist $\mathrm{DEC}(\varphi)$ vollständig in $\Sigma_i^p$. Insbesondere ist $\mathrm{DEC}(\varphi)$ für $\varphi \in \Sigma_1^{FO}$* NP*-vollständig.*
*2. Für $\varphi \in \Pi_i^{FO}$ ist $\mathrm{DEC}(\varphi)$ vollständig in $\Pi_i^p$. Insbesondere ist $\mathrm{DEC}(\varphi)$ für $\varphi \in \Pi_1^{FO}$* coNP*-vollständig.*
*3. Für $\varphi \in FO$ ist $\mathrm{DEC}(\varphi)$ vollständig in* PSPACE*.* **ohne Beweis**

Eine der interessantesten Verbindungen zwischen der Logik und der Komplexitätstheorie ist der Satz von Fagin (siehe Satz 5.40 unten), einem der zentralen Ergebnisse der in Abschnitt 4.8 erwähnten darstellenden Komplexitätstheorie. Die Definition des Entscheidungsproblems $\mathrm{DEC}_\varphi$ findet man in Abschnitt 4.8, und für die Klassen $\Sigma_0^{\mathrm{SO}}$ und $\Pi_0^{\mathrm{SO}}$ von Formeln zweiter Ordnung ohne Quantifizierungen erinnern wir an Definition 4.48.

**Satz 5.40 (Fagin [Fag74]).** *Ein Entscheidungsproblem $\mathrm{DEC}_\varphi$ ist genau dann in $\Sigma_i^p$, wenn $\varphi$ eine Formel aus $\Sigma_i^{SO}$ ist. Insbesondere ist $\mathrm{DEC}_\varphi$ genau dann in der Komplexitätsklasse* NP*, wenn $\varphi$ eine Formel aus $\Sigma_1^{SO}$ ist.* **ohne Beweis**

Dieser Satz besagt nicht nur, dass NP-vollständige Probleme mit Formeln aus $\Sigma_1^{\mathrm{SO}}$ definiert werden können, sondern auch, dass alle Entscheidungsprobleme aus NP mit Formeln aus $\Sigma_1^{\mathrm{SO}}$ definiert werden können. Die Klasse NP ist also nicht nur der Abschluss der Klasse $\Sigma_1^{\mathrm{SO}}$ unter $\leq_{\mathrm{m}}^{\mathrm{p}}$-Reduktionen, sondern NP ist die Klasse $\Sigma_1^{\mathrm{SO}}$ selbst.

## 5.2 Parametrisierte Komplexitätstheorie

Wie kann man mit Problemen umgehen, die vermutlich nicht in Polynomialzeit lösbar sind? Viele dieser Probleme (wie z. B. die aus Abschnitt 3.4) sind nicht nur interessant, sondern auch in praktischen Anwendungen wichtig. Die Tatsache, dass sie NP-hart sind und sich folglich einer effizienten Lösbarkeit entziehen, verringert weder den Wunsch noch die Notwendigkeit, sie zu lösen. Es gibt verschiedene Ansätze, solchen störrischen Problemen beizukommen. Beispielsweise kann man sich mit einer Lösungsnäherung zufrieden geben (also einen effizienten Approximationsalgorithmus entwerfen), oder man nimmt Fehler bei der Lösung in Kauf (entwirft also einen effizienten randomisierten Algorithmus). Einen dritten solchen Ansatz für den Umgang mit NP-harten Problemen stellen wir in diesem Abschnitt vor.

Die Untersuchung der parametrisierten Komplexität von NP-harten Problemen wurde von Downey und Fellows in den 1990er Jahren angestoßen und ist einer der jüngsten Ansätze zur Handhabung schwerer Probleme. Die Hauptidee besteht in der Isolierung eines Parameters als Teil des Problems, auf den das exponentielle Verhalten beschränkt werden kann. Damit sind insbesondere für gewisse kleine Parametergrößen effiziente exakte Algorithmen für schwere Probleme möglich.

### 5.2.1 Parametrisierte Probleme, FPT und XP

Eine klassische Komplexitätsanalyse beschränkt sich auf z. B. die Laufzeit eines Problems. In der parametrisierten Komplexitätstheorie betrachtet man seine Parameter

separat und führt abhängig von den Parametern gewissermaßen eine mehrdimensionale Komplexitätsanalyse aus. Deshalb erweitern wir Entscheidungsprobleme zu parametrisierten Problemen.

**Definition 5.41 (parametrisiertes Problem).** *Ein* parametrisiertes Problem *ist ein Paar* $(\Pi, \kappa)$*, wobei* $\Pi$ *ein Entscheidungsproblem mit Instanzenmenge* $\mathscr{I}$ *und*

parametrisiertes Problem

$$\kappa : \mathscr{I} \to \mathbb{N},$$

*der so genannte* Parameter*, eine in Polynomialzeit berechenbare Funktion ist.*

Wir unterscheiden eine parametrisierte Version eines Entscheidungsproblems von dem zugrundeliegenden Entscheidungsproblem durch einen Zusatz in der Problembezeichnung, wie z. B. einem dem Problemnamen vorangestellten „$p$“. Die folgenden Beispiele geben unterschiedliche Möglichkeiten an, Parametrisierungen zu wählen.

**Beispiel 5.42 (parametrisiertes Problem).** Eine natürliche Parametrisierung $\kappa$ eines Entscheidungsproblems kann man häufig dadurch erhalten, dass man für $\kappa$ den in der Probleminstanz vorkommenden Parameter wählt. Beispielsweise ergibt sich so durch die Abbildung $\kappa(G,k) = k$ eine Parametrisierung für das Entscheidungsproblem KNOTENÜBERDECKUNG. Das entsprechende parametrisierte Problem (KNOTENÜBERDECKUNG,$\kappa$) bezeichnen wir mit $p$-KNOTENÜBERDECKUNG. In der Problembeschreibung gibt es nun zusätzlich zum „Gegeben“- und „Frage“-Feld auch ein „Parameter“-Feld.

$p$-KNOTEN-ÜBERDECKUNG

| $p$-KNOTENÜBERDECKUNG | |
|---|---|
| *Gegeben:* | Ein Graph $G$ und eine Zahl $k \in \mathbb{N}$. |
| *Parameter:* | $k$. |
| *Frage:* | Gibt es in $G$ eine Knotenüberdeckung der Größe höchstens $k$? |

**Beispiel 5.43 (parametrisiertes Problem).** Eine weitere Möglichkeit der Parametrisierung zeigt das Beispiel *p-deg*-UNABHÄNGIGE MENGE. Zur Erinnerung: Der Parameter $\Delta(G)$ bezeichnet den maximalen Knotengrad eines Graphen $G$ (siehe Abschnitt 3.1).

*p-deg*-UNABHÄN-GIGE MENGE

| *p-deg*-UNABHÄNGIGE MENGE | |
|---|---|
| *Gegeben:* | Ein Graph $G$ und eine Zahl $k \in \mathbb{N}$. |
| *Parameter:* | $k + \Delta(G)$. |
| *Frage:* | Gibt es in $G$ eine unabhängige Menge der Größe mindestens $k$? |

**Beispiel 5.44 (parametrisiertes Problem).** Eine dritte mögliche Parametrisierung, diesmal für das Entscheidungsproblem SAT, ist durch die Abbildung $\kappa$ gegeben, die einer booleschen Formel die Anzahl der in ihr vorkommenden Variablen zuordnet. Das parametrisierte Problem $(\mathrm{SAT}, \kappa)$ bezeichnen wir kurz mit $p$-SAT.

$p$-SAT

| $p$-SAT | |
|---|---|
| *Gegeben:* | Eine boolesche Formel $\varphi$ in konjunktiver Normalform. |
| *Parameter:* | Anzahl der Variablen in $\varphi$. |
| *Frage:* | Ist $\varphi$ erfüllbar? |

### Die Komplexitätsklasse FPT

Der für NP-harte Probleme anscheinend unvermeidliche exponentielle Zeitbedarf bei der Lösung durch einen deterministischen Algorithmus kann manchmal in einem Parameter eingefangen und dort – in gewissem Sinn – entschärft werden. Probleme, bei denen dies gelingt, nennt man fest-Parameter-berechenbar (englisch: *fixed-parameter tractable*) und fasst sie in der Komplexitätsklasse FPT zusammen.

**Definition 5.45 (FPT-Algorithmus, fest-Parameter-berechenbar und FPT).** *Es seien $(\Pi, \kappa)$ ein parametrisiertes Problem, $\mathscr{I}$ die Instanzenmenge von $\Pi$ und $\kappa : \mathscr{I} \to \mathbb{N}$ eine Parametrisierung.*

FPT-Algorithmus

1. *Ein Algorithmus $A$ heißt* FPT-Algorithmus bezüglich einer Parametrisierung $\kappa$*, falls es eine berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ und ein Polynom $p$ gibt, sodass für jede Instanz $I \in \mathscr{I}$ die Laufzeit von $A$ bei Eingabe von $I$ (mit Eingabengröße $|I|$) durch*
$$f(\kappa(I)) \cdot p(|I|) \tag{5.11}$$
*abgeschätzt werden kann. (Alternativ zu (5.11) sagt man auch, die Laufzeit eines FPT-Algorithmus ist in $f(\kappa(I)) \cdot |I|^{\mathscr{O}(1)}$.)*

fest-Parameter-berechenbar

2. *Ein parametrisiertes Problem $(\Pi, \kappa)$ heißt* fest-Parameter-berechenbar*, falls es einen FPT-Algorithmus bezüglich $\kappa$ gibt, der das Entscheidungsproblem $\Pi$ löst.*

FPT

3. FPT *ist die Menge aller parametrisierten Probleme, die fest-Parameter-berechenbar sind.*

**Anmerkung 5.46.** Offensichtlich gilt $(\Pi, \kappa) \in \mathrm{FPT}$ für jedes Entscheidungsproblem $\Pi \in \mathrm{P}$ in jeder Parametrisierung $\kappa$ von $\Pi$.

In der angegebenen Definition der Laufzeit von FPT-Algorithmen werden der „gute“ und der „böse“ Teil (also $p(|I|)$ und $f(\kappa(I))$) durch eine Multiplikation getrennt. Man kann zeigen, dass sich u. a. dann eine äquivalente Definition ergibt, wenn man statt der Mutiplikation eine Addition einsetzt.

**Lemma 5.47.** *Es sei $(\Pi, \kappa)$ ein parametrisiertes Problem. Die folgenden Aussagen sind äquivalent:*

1. $(\Pi, \kappa) \in$ FPT.
2. *Es gibt eine berechenbare Funktion* $f : \mathbb{N} \to \mathbb{N}$*, ein Polynom* $p$ *und einen Algorithmus, der* $\Pi$ *für eine beliebige Instanz* $I$ *in der Zeit*

$$f(\kappa(I)) + p(|I|)$$

*löst.*
3. *Es gibt berechenbare Funktionen* $f, g : \mathbb{N} \to \mathbb{N}$*, ein Polynom* $p$ *und einen Algorithmus, der* $\Pi$ *für eine beliebige Instanz* $I$ *in der Zeit*

$$g(\kappa(I)) + f(\kappa(I)) \cdot p(|I| + \kappa(I))$$

*löst.*

**ohne Beweis**

Als Beispiel für einen FPT-Algorithmus wollen wir die folgende Aussage zeigen.

**Satz 5.48.** *Das parametrisierte Problem* $p$-KNOTENÜBERDECKUNG *ist in* FPT.

**Beweis.** Wir geben einen Algorithmus an, der $p$-KNOTENÜBERDECKUNG in der Zeit $\mathcal{O}(2^k \cdot n)$ löst. Dieser Algorithmus folgt der in Abschnitt 3.5 beschriebenen Teile-und-herrsche-Strategie, um einen Suchbaum beschränkter Tiefe zu durchsuchen. Ein *Suchbaum* für eine Instanz $(I,k)$ eines parametrisierten Problems $(\Pi, \kappa)$ ist ein knotenmarkierter Wurzelbaum. Die Wurzel des Suchbaums wird mit der Instanz $(I,k)$ markiert. Die weiteren Knoten werden mit Instanzen $(I',k')$ für Teilprobleme rekursiv durch Anwendung problemspezifischer Verzweigungsregeln markiert, wobei der Parameterwert $k'$ der Kinder eines Knotens echt kleiner als der des Knotens selbst ist und nicht negativ wird. Eine Instanz zu einem Knoten $u$ im Suchbaum ist genau dann erfüllbar, wenn mindestens eine der Instanzen der Kinder von $u$ erfüllbar ist. Die *Größe eines Suchbaums* ist die Anzahl seiner Knoten.

Suchbaum

Größe eines Suchbaums

Der Algorithmus beruht auf der folgenden einfachen Idee. Es sei $(G,k)$ eine Instanz von $p$-KNOTENÜBERDECKUNG. Wir wählen eine beliebige Kante $\{v_1, v_2\}$ aus $G$. Folglich hat $G$ genau dann eine Knotenüberdeckung der Größe höchstens $k$, wenn $G - v_1$ eine Knotenüberdeckung der Größe höchstens $k-1$ hat oder $G - v_2$ eine Knotenüberdeckung der Größe höchstens $k-1$ hat, wobei $G - v_i$, $i \in \{1,2\}$, den Teilgraphen von $G$ ohne den Knoten $v_i$ und die zu $v_i$ inzidenten Kanten bezeichnet. Demgemäß nutzt unser Suchbaum die folgende Verzweigungsregel:

V0 Wähle eine Kante $\{v_1, v_2\}$ und füge entweder den Knoten $v_1$ oder den Knoten $v_2$ zur bisher konstruierten Knotenüberdeckung hinzu.

Der Algorithmus arbeitet bei Eingabe einer Instanz $(G,k)$ von $p$-KNOTENÜBERDECKUNG so:

1. Konstruiere einen vollständigen binären Wurzelbaum $T$ der Höhe $k$.
2. Markiere den Wurzelknoten in $T$ mit der gegebenen Probleminstanz $(G,k)$.

3. Die vom Wurzelknoten verschiedenen Knoten in $T$ werden in Abhängigkeit von der Markierung ihrer Vorgängerknoten wie folgt markiert. Es sei $u$ ein mit $(G',k')$ markierter innerer Baumknoten, dessen beide Kinder noch unmarkiert sind. Hat $G' = (V',E')$ noch mindestens eine Kante (d. h., $E' \neq \emptyset$), so wähle eine beliebige Kante $\{v_1, v_2\}$ aus $G'$ und
   a) markiere ein Kind von $u$ mit $(G' - v_1, k' - 1)$;
   b) markiere das andere Kind von $u$ mit $(G' - v_2, k' - 1)$.
4. Falls es einen Baumknoten gibt, der mit $(G',k')$ markiert ist, wobei $G' = (V',\emptyset)$ gilt, so hat $(G,k)$ eine Knotenüberdeckung der Größe höchstens $k$; andernfalls gibt es keine Knotenüberdeckung der Größe höchstens $k$ für $G$.

Die Korrektheit des Algorithmus ist offensichtlich, und seine Laufzeit $\mathcal{O}(2^k \cdot n)$ ist ebenfalls leicht zu sehen. Die beiden Instanzen $(G' - v_i, k-1)$, $i \in \{1,2\}$, können offenbar in der Zeit $\mathcal{O}(n)$ aus $G'$ konstruiert werden. Weiterhin kann für jeden Graphen $G' = (V,\emptyset)$ die Instanz $(G',k')$ für das Problem $p$-KNOTENÜBERDECKUNG in der Zeit $\mathcal{O}(n)$ gelöst werden. Da der Suchbaum eine Höhe von $k$ hat, müssen $2^k$ Teilprobleme gelöst werden, und damit ist die Laufzeit insgesamt in $\mathcal{O}(2^k \cdot n)$. ❑

In Abb. 5.10 ist ein Suchbaum für einen Gittergraphen angedeutet.

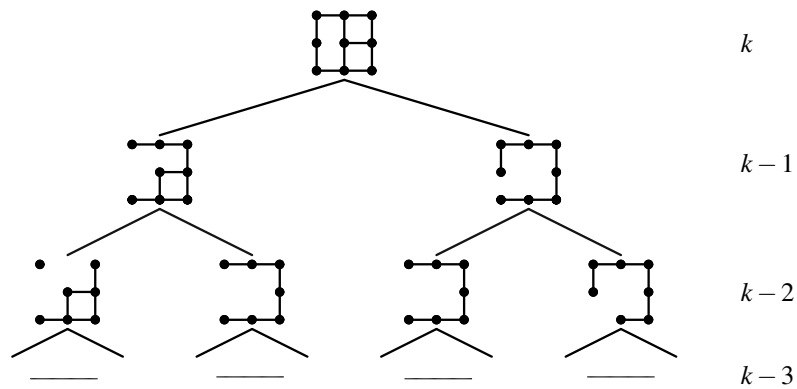



**Abb. 5.10.** Suchbaum für $p$-KNOTENÜBERDECKUNG in einem Gittergraphen

**Übung 5.49.** Vervollständigen Sie den Suchbaum aus Abb. 5.10 für $p$-KNOTENÜBERDECKUNG für den Gittergraph $G_{3,3}$ und $k = 3$.

LOG KNOTENÜBERDECKUNG

Wir betrachten nun die folgende Variante des Knotenüberdeckungsproblems:

| LOG KNOTENÜBERDECKUNG | |
|---|---|
| *Gegeben:* | Ein Graph $G = (V,E)$. |
| *Frage:* | Hat $G$ eine Knotenüberdeckung der Größe höchstens $\log(\|V\|)$? |

**Korollar 5.50.** *Das Problem* LOG KNOTENÜBERDECKUNG *kann für jeden Graphen $G$ mit $n$ Knoten in der Zeit $\mathcal{O}(n^2)$ gelöst werden.*

Der Beweis der folgenden Aussage ist einfach.

**Satz 5.51.** $p$-SAT $\in$ FPT. **ohne Beweis**

**Übung 5.52.** (a) Beweisen Sie Satz 5.51.
(b) Es seien $\Pi$ ein Entscheidungsproblem mit der Instanzenmenge $\mathscr{I}$ und $\kappa_{\text{one}} : \mathscr{I} \to \mathbb{N}$ mit $\kappa_{\text{one}}(I) = 1$ eine Parametrisierung für $\Pi$.
Zeigen Sie, dass das Entscheidungsproblem $\Pi$ genau dann in Polynomialzeit lösbar ist, wenn das parametrisierte Problem $(\Pi, \kappa_{\text{one}})$ fest-Parameter-berechenbar ist.

Um die Nichtzugehörigkeit eines parametrisierten Problems zur Klasse FPT zu zeigen (unter der vernünftigen Annahme P $\neq$ NP, siehe Abschnitt 5.1.3), benötigen wir die folgende Definition.

**Definition 5.53 ($k$-te Scheibe).** *Es seien $(\Pi, \kappa)$ ein parametrisiertes Problem und $k \in \mathbb{N}$. Die $k$-te* Scheibe *(englisch:* the $k$th slice*)* von $(\Pi, \kappa)$ *ist das (klassische) Entscheidungsproblem $\Pi$, in dem für jede Instanz $I$ die Einschränkung $\kappa(I) = k$ gilt.*

$k$-te Scheibe

**Beispiel 5.54 ($k$-te Scheibe).** Wir betrachten das parametrisierte Problem $p$-PARTITION IN UNABHÄNGIGE MENGEN:

$p$-PARTITION IN UNABHÄNGIGE MENGEN

| $p$-PARTITION IN UNABHÄNGIGE MENGEN | |
|---|---|
| *Gegeben:* | Ein Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$. |
| *Parameter:* | $k$. |
| *Frage:* | Gibt es eine Partition von $V$ in $k$ unabhängige Mengen? |

Die dritte Scheibe dieses parametrisierten Problems ist das Entscheidungsproblem PARTITION IN DREI UNABHÄNGIGE MENGEN :

PARTITION IN DREI UNABHÄNGIGE MENGEN

| PARTITION IN DREI UNABHÄNGIGE MENGEN | |
|---|---|
| *Gegeben:* | Ein Graph $G = (V, E)$. |
| *Frage:* | Gibt es eine Partition von $V$ in drei unabhängige Mengen? |

das uns auch unter dem Namen 3-FÄRBBARKEIT bekannt ist, siehe Abschnitt 3.4.2.

3-FÄRBBARKEIT

**Satz 5.55.** *Es seien $(\Pi, \kappa)$ ein parametrisiertes Problem und $k \in \mathbb{N}$. Ist $(\Pi, \kappa)$ fest-Parameter-berechenbar, so ist die $k$-te Scheibe von $(\Pi, \kappa)$ in Polynomialzeit lösbar.*

**Beweis.** Da $(\Pi, \kappa)$ fest-Parameter-berechenbar ist, gibt es einen FPT-Algorithmus $A$, der $\Pi$ löst. Somit gibt es eine berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ und ein Polynom $p$, sodass für jede Instanz $I \in \mathscr{I}$ die Laufzeit von $A$ bei Eingabe von $I$ durch $f(\kappa(I)) \cdot |I|^{\mathscr{O}(1)}$ abgeschätzt werden kann. Da in der $k$-ten Scheibe von $(\Pi, \kappa)$ für alle Instanzen $I$ nun $\kappa(I) = k$ gilt, entartet $f$ zu der konstanten Funktion $f(k)$. Folglich kann die Laufzeit von $A$ durch $f(k) \cdot |I|^{\mathscr{O}(1)} \in \mathscr{O}(|I|^{\mathscr{O}(1)})$ abgeschätzt werden und ist somit polynomiell in der Instanzengröße $|I|$. ❑

**Korollar 5.56.** *Ist* P $\neq$ NP*, so ist* $p$-PARTITION IN UNABHÄNGIGE MENGEN $\notin$ FPT.

**Beweis.** Angenommen, das Problem $p$-PARTITION IN UNABHÄNGIGE MENGEN wäre in der Klasse FPT. Dann ist nach Satz 5.55 insbesondere die dritte Scheibe von $p$-PARTITION IN UNABHÄNGIGE MENGEN in Polynomialzeit lösbar. Da jedoch nach Satz 5.26 das Entscheidungsproblem PARTITION IN DREI UNABHÄNGIGE MENGEN (alias 3-FÄRBBARKEIT) NP-vollständig ist, folgt nach der dritten Aussage von Satz 5.19, dass P = NP gilt. Dies ist ein Widerspruch zur Voraussetzung P $\neq$ NP. Folglich ist die Annahme falsch und die Behauptung gezeigt. ❑

**Übung 5.57.** Erläutern Sie, warum die erste Scheibe und die zweite Scheibe des parametrisierten Problems $p$-PARTITION IN UNABHÄNGIGE MENGEN in Polynomialzeit lösbar sind.

$p$-UNABHÄNGIGE MENGE

Leider gibt es für viele der parametrisierten Probleme, die vermutlich nicht in FPT liegen, keine solchen Bezüge zur klassischen Komplexitätstheorie. Zum Beispiel ist das parametrisierte Problem $p$-UNABHÄNGIGE MENGE:

| $p$-UNABHÄNGIGE MENGE | |
|---|---|
| *Gegeben:* | Ein Graph $G$ und eine Zahl $k \in \mathbb{N}$. |
| *Parameter:* | $k$. |
| *Frage:* | Gibt es in $G$ eine unabhängige Menge der Größe mindestens $k$? |

$p$-CLIQUE

vermutlich nicht in der Menge FPT enthalten und dennoch sind alle Scheiben dieses Problems in P (siehe Übung 5.58). Analoge Aussagen gelten für die parametrisierten Probleme $p$-CLIQUE und $p$-DOMINIERENDE MENGE, die wie folgt definiert sind:

| $p$-CLIQUE | |
|---|---|
| *Gegeben:* | Ein Graph $G$ und eine Zahl $k \in \mathbb{N}$. |
| *Parameter:* | $k$. |
| *Frage:* | Gibt es in $G$ eine Clique der Größe mindestens $k$? |

| $p$-DOMINIERENDE MENGE | |
|---|---|
| *Gegeben:* | Ein Graph $G$ und eine Zahl $k \in \mathbb{N}$. |
| *Parameter:* | $k$. |
| *Frage:* | Gibt es in $G$ eine dominierende Menge der Größe höchstens $k$? |

$p$-DOMINIERENDE MENGE

**Übung 5.58.** Zeigen Sie, dass alle Scheiben des parametrisierten Problems $p$-UNABHÄNGIGE MENGE in der Klasse P enthalten sind.

**Anmerkung 5.59.** Die Probleme $p$-CLIQUE, $p$-UNABHÄNGIGE MENGE und $p$-DOMINIERENDE MENGE liegen vermutlich nicht in der Klasse FPT. Wir werden im folgenden Abschnitt zeigen, dass diese Probleme (bezüglich der in Definition 5.67 unten eingeführten $\leq_m^{\text{fpt}}$-Reduzierbarkeit) vollständig in parametrisierten Komplexitätsklassen sind, die vermutlich größer als FPT sind.

Natürlich ist die Zugehörigkeit eines Problems zur Klasse FPT von der gewählten Parametrisierung abhängig. Der folgende Satz gibt ohne Beweis ein Beispiel an.

**Satz 5.60.** *$p$-deg-*UNABHÄNGIGE MENGE *ist in* FPT. **ohne Beweis**

In den Kapiteln 10 und 11 werden wir weitere Parametrisierungen betrachten.

### Die Komplexitätsklasse XP

In der parametrisierten Komplexitätstheorie unterscheidet man zwischen Laufzeiten wie $\mathcal{O}(2^{\kappa(I)} \cdot |I|)$ (fest-Parameter-berechenbar) und $\mathcal{O}(|I|^{\kappa(I)})$ (genügt nicht zum Nachweis der Fest-Parameter-Berechenbarkeit). Laufzeiten vom letzteren Typ wollen wir nun genauer betrachten.

**Definition 5.61 (XP-Algorithmus und XP).**

1. *Es seien $(\Pi, \kappa)$ ein parametrisiertes Problem und $\mathcal{I}$ die Instanzenmenge von $\Pi$. Ein Algorithmus $A$ heißt* XP-Algorithmus bezüglich einer Parametrisierung $\kappa : \mathcal{I} \to \mathbb{N}$, *falls es berechenbare Funktionen $f, g : \mathbb{N} \to \mathbb{N}$ gibt, sodass für jede Instanz $I \in \mathcal{I}$ die Laufzeit von $A$ bei Eingabe von $I$ (mit Eingabengröße $|I|$) durch*
$$f(\kappa(I)) \cdot |I|^{g(\kappa(I))}$$
*abgeschätzt werden kann.* XP-Algorithmus
2. *Die Menge aller parametrisierten Probleme, die durch einen XP-Algorithmus gelöst werden können, fasst man in der Komplexitätsklasse* XP *zusammen.* XP

**Anmerkung 5.62.** Es gilt offensichtlich die Beziehung:
$$\text{FPT} \subseteq \text{XP}.$$

**Beispiel 5.63.** Die parametrisierten Probleme $p$-CLIQUE, $p$-UNABHÄNGIGE MENGE und $p$-DOMINIERENDE MENGE liegen in der Klasse XP (siehe Übung 5.71).

### 5.2.2 W-Hierarchie

Wir definieren nun parametrisierte Komplexitätsklassen zwischen FPT und XP. Diese Klassen bilden die so genannte W-Hierarchie, deren Stufen in der parametrisierten Komplexitätstheorie eine ähnliche Rolle wie die Klasse NP und die höheren Stufen der Polynomialzeit-Hierarchie[31] in der klassischen Komplexitätstheorie spielen: Probleme, die bezüglich einer geeigneten parametrisierten Reduzierbarkeit (siehe Definition 5.67 unten) vollständig für die Stufen $\mathrm{W}[t]$, $t \geq 1$, der W-Hierarchie sind, sind vermutlich nicht fest-Parameter-berechenbar.

In der W-Hierarchie klassifiziert man Probleme bezüglich ihrer Tiefe in einem geeigneten booleschen Schaltkreis (mit nur einem Ausgabegatter). Boolesche Schaltkreise wurden in Definition 4.14 eingeführt; das folgende Beispiel soll diese in Erinnerung rufen.

**Beispiel 5.64 (boolescher Schaltkreis).** Abbildung 5.11 zeigt einen booleschen Schaltkreis mit fünf Eingängen (wie wir Eingabegatter auch nennen wollen) und einem Ausgang (d. h. Ausgabegatter).

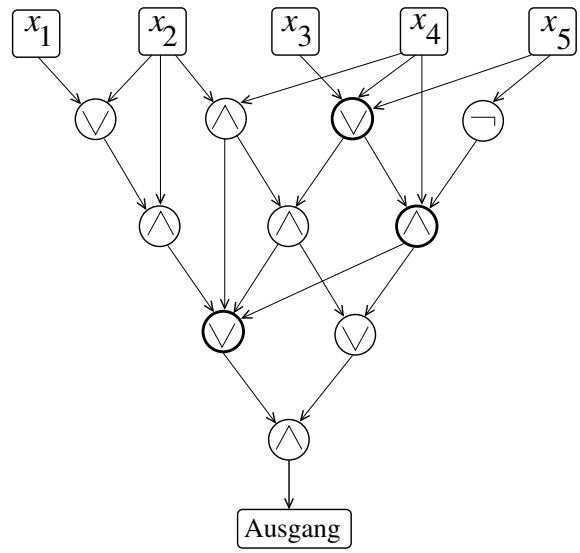



**Abb. 5.11.** Ein boolescher Schaltkreis mit fünf Eingängen und einem Ausgang

Werden die Eingänge des Schaltkreises in Abb. 5.11 etwa mit den Werten

$$(x_1, x_2, x_3, x_4, x_5) = (0, 1, 1, 0, 1)$$

belegt, so ergibt sich am Ausgang der Wert 0.

[31] Die Polynomialzeit-Hierarchie wurde kurz in Abschnitt 5.1.4 und wird ausführlich z. B. in [Rot08] behandelt.

**Definition 5.65 (großes Gatter, Weft und Höhe).**

1. *Gatter in einem booleschen Schaltkreis, die einen Eingangsgrad größer als zwei haben, werden als* große Gatter *bezeichnet.* — großes Gatter
2. *Die* Weft eines booleschen Schaltkreises *ist die maximale Anzahl an großen Gattern auf einem gerichteten Weg von einem Eingang zum Ausgang.* — Weft
3. *Die* Höhe eines booleschen Schaltkreises *ist die Länge eines längsten gerichteten Wegs von einem Eingang zum Ausgang.* — Höhe

**Beispiel 5.66 (großes Gatter, Weft und Höhe).** In Abb. 5.11 sind die großen Gatter durch stärker umrandete Kreise markiert. Der dargestellte Schaltkreis hat die Weft 3 und die Höhe 5.

**Definition 5.67 (parametrisierte Reduktion).** *Ein parametrisiertes Problem* $(\Pi_1, \kappa_1)$ *mit Instanzenmenge* $\mathscr{I}_1$ *lässt sich auf ein parametrisiertes Problem* $(\Pi_2, \kappa_2)$ *mit Instanzenmenge* $\mathscr{I}_2$ parametrisiert reduzieren *(kurz* $(\Pi_1, \kappa_1) \leq_m^{\text{fpt}} (\Pi_2, \kappa_2)$*), wenn es eine Funktion* $r : \mathscr{I}_1 \to \mathscr{I}_2$ *gibt, sodass gilt:* — parametrisierte Reduzierbarkeit $\leq_m^{\text{fpt}}$

1. *für alle* $I \in \mathscr{I}_1$ *ist* $I$ *genau dann eine „Ja"-Instanz von* $\Pi_1$*, wenn* $r(I)$ *eine „Ja"-Instanz von* $\Pi_2$ *ist,*
2. $r(I)$ *ist für eine berechenbare Funktion* $f$ *und ein Polynom* $p$ *in der Zeit*

$$f(\kappa_1(I)) \cdot p(|I|)$$

   *berechenbar und*
3. *für* $I \in \mathscr{I}_1$ *und eine berechenbare Funktion* $g : \mathbb{N} \to \mathbb{N}$ *gilt* $\kappa_2(r(I)) \leq g(\kappa_1(I))$.

Die ersten beiden Bedingungen in der Definition der parametrisierten Reduzierbarkeit ähneln denen der (klassischen) $\leq_m^p$-Reduzierbarkeit aus Definition 5.18. Die dritte Bedingung in Definition 5.67 besagt, dass der neue Parameter nur vom alten Parameter abhängen darf und *nicht* von der Instanzengröße.

Die aus NP-Vollständigkeitsbeweisen (siehe Abschnitt 5.1.3) bekannten klassischen Reduktionen sind im Allgemeinen keine parametrisierten Reduktionen, wenn man die Probleme als parametrisierte Probleme definiert.

**Beispiel 5.68 (parametrisierte Reduktion).**

1. Die klassische Reduktion (siehe z. B. Garey und Johnson [GJ79])

$$\text{UNABHÄNGIGE MENGE} \leq_m^p \text{KNOTENÜBERDECKUNG}$$

   bildet eine Instanz $(G,k)$ von UNABHÄNGIGE MENGE, wobei $G = (V,E)$ ein Graph ist, auf eine Instanz $(G, |V| - k)$ von KNOTENÜBERDECKUNG ab, d. h., hier wird der Parameter $\kappa(I) = k$ in Abhängigkeit von der Instanzengröße modifiziert, was bei parametrisierten Reduktionen nicht erlaubt ist.
2. In der klassischen Reduktion (siehe z. B. Garey und Johnson [GJ79])

$$\text{CLIQUE} \leq_m^p \text{UNABHÄNGIGE MENGE}$$

   hingegen ersetzt man $(G,k)$ durch $(\overline{G},k)$, wobei $\overline{G}$ der Komplementgraph von $G$ ist, d. h., hier wird der Parameter $\kappa(I) = k$ nicht abhängig von der Instanzengröße modifiziert. Somit ist diese Reduktion eine parametrisierte Reduktion.

Wir definieren nun das gewichtete Erfüllbarkeitsproblem, welches in der parametrisierten Komplexitätstheorie gewissermaßen die Rolle des Erfüllbarkeitsproblems SAT in der klassischen Komplexitätstheorie spielt.

*p*-WEIGHTED SAT$(t,h)$

| *p*-WEIGHTED SAT$(t,h)$ | |
|---|---|
| *Gegeben:* | Ein boolescher Schaltkreis $C$ mit Weft $t$ und Höhe $h$ und eine Zahl $k \in \mathbb{N}$. |
| *Parameter:* | $k$. |
| *Frage:* | Gibt es eine erfüllende Belegung von $C$ mit Gewicht $k$? |

Dabei ist das Gewicht einer erfüllenden Belegung die Anzahl der Variablen, die mit dem Wahrheitswert 1 belegt werden.

W-Hierarchie
W$[t]$

**Definition 5.69 (W-Hierarchie).** *Die* W-Hierarchie *besteht aus den Komplexitätsklassen* W$[t]$, $t \geq 1$. *Ein parametrisiertes Problem* $(\Pi,\kappa)$ *liegt in der Klasse* W$[t]$, *wenn es mittels einer parametrisierten Reduktion auf* *p*-WEIGHTED SAT$(t,h)$ *für eine natürliche Zahl h reduziert werden kann, d. h.:*

$$\mathrm{W}[t] = \{(\Pi,\kappa) \mid (\Pi,\kappa) \leq_{\mathrm{m}}^{\mathrm{fpt}} \textit{p}\text{-WEIGHTED SAT}(t,h) \textit{ für ein } h \in \mathbb{N}\}.$$

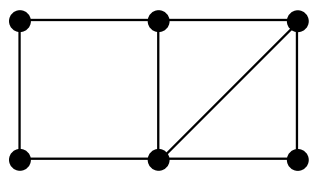
**Beispiel 5.70 (Probleme in W[1] und W[2]).**

1. *p*-UNABHÄNGIGE MENGE $\in$ W$[1]$: Ein Graph $G = (V,E)$ mit Knotenmenge $V = \{1,\ldots,n\}$ hat genau dann eine unabhängige Menge der Größe $k$, wenn die Formel
$$\bigwedge_{\{i,j\}\in E} (\neg x_i \vee \neg x_j) \tag{5.12}$$
über den booleschen Variablen $x_1,\ldots,x_n$ eine erfüllende Belegung mit Gewicht $k$ hat.
Die Formel (5.12) ist einfach zu verstehen, wenn man sich klar macht, dass für jede Kante $\{i,j\} \in E$ des Graphen höchstens einer der beiden Endknoten in einer unabhängigen Menge liegen kann. Dies entspricht der logischen Formel $\neg(x_i \wedge x_j)$, die semantisch äquivalent zu der Formel $\neg x_i \vee \neg x_j$ ist.
Da die Formel (5.12) in 2-KNF ist, kann sie durch einen booleschen Schaltkreis mit Weft 1 und Höhe 3 beschrieben werden, und somit ist *p*-UNABHÄNGIGE MENGE auf *p*-WEIGHTED SAT$(1,3)$ parametrisiert reduzierbar. Folglich liegt *p*-UNABHÄNGIGE MENGE in der Klasse W$[1]$.
2. *p*-CLIQUE $\in$ W$[1]$: Da nach Beispiel 5.68 *p*-CLIQUE $\leq_{\mathrm{m}}^{\mathrm{fpt}}$ *p*-UNABHÄNGIGE MENGE gilt, *p*-UNABHÄNGIGE MENGE in W$[1]$ liegt und die Klasse W$[1]$ unter $\leq_{\mathrm{m}}^{\mathrm{fpt}}$-Reduktionen abgeschlossen ist, ist auch *p*-CLIQUE in W$[1]$.
3. *p*-DOMINIERENDE MENGE $\in$ W$[2]$: Ein Graph $G = (V,E)$ mit $V = \{1,\ldots,n\}$ hat genau dann eine dominierende Menge der Größe $k$, wenn die Formel
$$\bigwedge_{i\in V} \bigvee_{j\in V:\{i,j\}\in E} (x_i \vee x_j) \tag{5.13}$$

über den booleschen Variablen $x_1, \ldots, x_n$ eine erfüllende Belegung mit Gewicht $k$ hat.
Die Formel (5.13) drückt aus, dass für jeden Knoten $i \in V$ des Graphen mindestens ein Nachbarknoten oder der Knoten selbst in der dominierenden Menge liegen muss.
Die Formel kann durch einen booleschen Schaltkreis mit Weft 2 und Höhe 3 beschrieben werden. Somit ist $p$-DOMINIERENDE MENGE auf $p$-WEIGHTED SAT$(2,3)$ parametrisiert reduzierbar. Also liegt $p$-DOMINIERENDE MENGE in der Klasse W[2].

**Übung 5.71.** In Beispiel 5.70 wurde gezeigt, dass $p$-CLIQUE, $p$-UNABHÄNGIGE MENGE und $p$-DOMINIERENDE MENGE in W[1] bzw. in W[2] und somit auch in XP liegen. Geben Sie explizite XP-Algorithmen für diese parametrisierten Probleme an.

**Übung 5.72.** Zeigen Sie (wie in Beispiel 5.70), dass $p$-KNOTENÜBERDECKUNG in der Klasse W[1] enthalten ist. Zeichnen Sie für den Graphen aus Abb. 3.9:



und Ihre boolesche Formel den zugehörigen booleschen Schaltkreis. Wie viele große Gatter benötigen Sie? Welche Werte haben die Weft und die Höhe Ihres booleschen Schaltkreises?

**Anmerkung 5.73.** Es gilt die folgende Inklusionskette der parametrisierten Komplexitätsklassen:

$$\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \text{W}[3] \subseteq \cdots \subseteq \text{XP}.$$

Man vermutet, dass die Inklusionen alle echt sind, auch wenn bisher kein Beweis dafür erbracht werden konnte.

**Definition 5.74 (Härte und Vollständigkeit für die Stufen der W-Hierarchie).**

1. *Ein parametrisiertes Problem* $(\Pi, \kappa)$ *ist* W[$t$]-hart, *wenn jedes Problem aus* W[$t$] *parametrisiert auf* $(\Pi, \kappa)$ *reduziert werden kann.* W[$t$]-hart
2. *Ein parametrisiertes Problem ist* W[$t$]-vollständig , *falls es in der Klasse* W[$t$] *liegt und* W[$t$]*-hart ist.* W[$t$]-vollständig

In der parametrisierten Komplexitätstheorie gibt es auch eine Aussage, die zum dritten Teil von Satz 5.19 in der klassischen Komplexitätstheorie analog ist.

**Satz 5.75.** *Für jedes* $t \geq 1$ *gilt* W[$t$] = FPT *genau dann, wenn ein* W[$t$]*-hartes Problem in* FPT *liegt.* **ohne Beweis**

**Übung 5.76.** Beweisen Sie Satz 5.75.
**Hinweis:** Der Beweis ist einfach, da die Klasse FPT unter $\leq_{\mathrm{m}}^{\mathrm{fpt}}$-Reduktionen abgeschlossen ist, d. h., gilt $A \leq_{\mathrm{m}}^{\mathrm{fpt}} B$ und $B \in \mathrm{FPT}$, so ist $A \in \mathrm{FPT}$.

Schließlich geben wir noch einige Vollständigkeitsaussagen ohne Beweise an.

**Satz 5.77.** *1. p-*CLIQUE *ist* W[1]*-vollständig.*
*2. p-*UNABHÄNGIGE MENGE *ist* W[1]*-vollständig.*
*3. p-*DOMINIERENDE MENGE *ist* W[2]*-vollständig.*

**ohne Beweis**

## 5.3 Literaturhinweise

Hartmanis, Lewis und Stearns [HS65, SHL65, HLS65, LSH65] legten die Fundamente der Komplexitätstheorie. Insbesondere führten sie die heute üblichen mathematischen Modelle zur Beschreibung der Komplexitätsmaße Zeit und Platz ein, und sie bewiesen die Sätze über lineare Platzkompression und Beschleunigung sowie die Hierarchiesätze für Zeit und Platz. Stearns verfasste eine interessante Beschreibung dieser aufregenden frühen Jahre der Komplexitätstheorie [Ste90].

Cobham [Cob64] und Edmonds [Edm65] verdanken wir die Erkenntnis, dass der informale Begriff der „effizienten“ oder „machbaren“ Berechnung am sinnvollsten durch die Klasse P formalisiert wird.

Der Satz von Cook stammt aus dem Jahr 1971 [Coo71]. Dieselbe Aussage (Satz 5.21) wurde unabhängig von Levin [Lev73] bewiesen. Dieses bemerkenswerte Resultat markiert den Beginn der Theorie der NP-Vollständigkeit, die sich später ausgesprochen gut entwickelt und eine außerordentlich reiche Literatur hervorgebracht hat und immer noch hervorbringt. Namentlich Karp [Kar72] trieb diese Richtung innerhalb der Komplexitätstheorie energisch voran. Seine Techniken gehören heute zum Standardrepertoire des Komplexitätstheoretikers. Lemma 5.24 und Satz 5.26 gehen auf Schaefer zurück [Sch78]. Einen umfangreichen Überblick über NP-vollständige Probleme liefert das Werk von Garey und Johnson [GJ79].

Die Polynomialzeit-Hierarchie wurde von Meyer und Stockmeyer [MS72, Sto77] eingeführt. Eine Vielzahl natürlicher vollständiger Probleme in den Stufen dieser Hierarchie wurden beispielsweise von Wrathall, Umans und Schaefer identifiziert [Wra77, Uma01, SU02a, SU02b].

Das in Abschnitt 5.1.4 erwähnte Resultat, dass das UNIQUE OPTIMAL TRAVELING SALESPERSON PROBLEM vollständig in $\mathrm{P}^{\mathrm{NP}}$ ist, geht auf Papadimitriou [Pap84] zurück. Krentel [Kre88] und Wagner [Wag87] sind viele $\mathrm{P}^{\mathrm{NP}}$-Vollständigkeitsresultate für Varianten von NP-Optimierungsproblemen wie ODD-MAX-SAT zu verdanken. Khuller und Vazirani [KV91] untersuchten das in Übung 5.30 betrachtete funktionale Problem, die bezüglich einer lexikographischen Ordnung der Farben kleinste Vierfärbung eines gegebenen planaren Graphen zu bestimmen. Sie zeigten,

dass dieses Problem in dem Sinn „NP-hart“ ist,[32] dass, wenn es in Polynomialzeit lösbar wäre, auch das NP-vollständige Dreifärbbarkeitsproblem für planare Graphen (siehe Stockmeyer [Sto73]) in Polynomialzeit gelöst werden könnte. Große, Rothe und Wechsung [GRW06] verbesserten dieses Resultat optimal und zeigten, dass dieses Problem in diesem Sinn sogar „$P^{NP}$-hart“ ist.

Es gibt viele empfehlenswerte Bücher zur Komplexitätstheorie, zum Beispiel die von Papadimitriou [Pap94], Balcázar, Díaz und Gabarró [BDG95, BDG90] und Bovet und Crescenzi [BC93]. Andere Bücher behandeln auch algorithmische Aspekte und Anwendungsgebiete der Komplexitätstheorie (wie z. B. die Kryptologie), siehe zum Beispiel [Rot08] bzw. die englische Urfassung [Rot05].

Das Gebiet der parametrisierten Komplexitätstheorie wurde von Downey und Fellows [DF95] begründet, die insbesondere die Klasse FPT und die W-Hierarchie einführten und untersuchten. Downey und Fellows [DF99], Flum und Grohe [FG06] und Niedermeier [Nie06] verfassten weiterführende Lehrbücher zur parametrisierten Komplexität. Der parametrisierte Algorithmus für Satz 5.48 wurde von Downey und Fellows vorgestellt [DF92] (siehe auch [DF99]). Die Beweise der Aussagen in Lemma 5.47 und Satz 5.77 findet man zum Beispiel im Buch von Flum und Grohe [FG06].

[32] NP-Härte im Sinne von Definition 5.18 bezieht sich auf Reduktionen zwischen Entscheidungsproblemen. Für Suchprobleme wie das in Übung 5.30 betrachtete Problem hingegen (die ja nicht nur eine Ja/Nein-Entscheidung treffen, sondern explizit eine Lösung ausgeben) verwendet man andere Arten von Reduzierbarkeit, wie zum Beispiel die von Krentel [Kre88] eingeführte *metrische Reduzierbarkeit*, die wir hier nicht formal definieren.

metrische Reduzierbarkeit