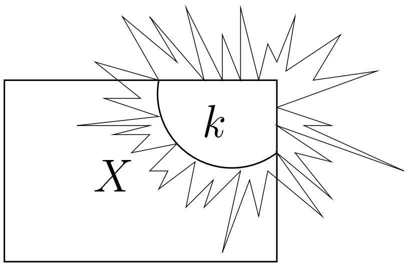
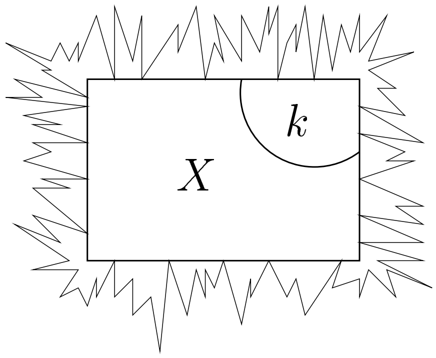
Parametrisierte Algorithmen *

Rolf Niedermeier
in Zusammenarbeit mit Jochen Alber
Wilhelm-Schickard-Institut für Informatik, Universität Tübingen,
Sand 13, D-72076 Tübingen
niedermr@informatik.uni-tuebingen.de



statt



*Skript zu einer zweistündigen Vorlesung im Sommersemester 1999 für den Bereich „Theoretische Informatik“ an der Universität Tübingen. Das Skript ist erhältlich über http://www-fs.informatik.uni-tuebingen.de/∼niedermr/lehre/
Version vom 2. September 1999.

Inhaltsverzeichnis

0 Vorwort

Viele Probleme von großer praktischer Bedeutung erweisen sich als NP-hart, das heißt, für sie sind keine effizienten Algorithmen bekannt. In der Praxis wird zu ihrer Lösung daher meist auf heuristische Verfahren zurückgegriffen, die zwar oftmals ausreichend gute Laufzeiten bzw. Lösungen liefern, aber leider meist schwer durchschaubar sind und keine garantierten Aussagen über ihre Leistungsgüte erlauben. Ein möglicher Ausweg aus dem „Dilemma der NP-Härte“ kann in der Betrachtung von „parametrisierter Komplexität“ bestehen: Bei vielen NP-harten Problemen läßt sich die scheinbar inhärente „kombinatorische Explosion“ auf einen kleinen Teil der Eingabe, einen sogenannten Parameter beschränken. Dies führt zu dem Konzept der parametrisierten Algorithmen, welche eine sinnvolle Alternative zu heuristischen Methoden darstellen können. In der Vorlesung werden die Möglichkeiten und Grenzen parametrisierter Algorithmen aufgezeigt.

Das Schwergewicht der Vorlesung liegt auf grundlegenden Methoden zur Entwicklung effizienter, parametrisierter Algorithmen. Diese werden an wichtigen Beispielen erläutert. Abgerundet wird diese Einführung in die Welt der parametrisierten Algorithmen mit einer kurzen Darstellung einer parametrisierten Komplexitätstheorie.

Die Hauptquelle zu dieser Vorlesung ist die jüngst erschienene Monographie von Downey and Fellows [12]. Es werden aber noch einige andere Übersichtsartikel [13, 20, 24] und Forschungspapiere herangezogen — die spezifischen Referenzen werden an den entsprechenden Textstellen gegeben und finden sich im Literaturverzeichnis am Ende des Skripts.

Zuletzt sein noch darauf verwiesen, daß der englischsprachige Fachbegriff genauer „fixed parameter algorithm“ lautet. Dieser wurde hier nur kurz mit „parametrisierter Algorithmus“ übersetzt.

Für diese Vorlesung setzen wir elementare Vorkenntnisse aus dem Grundstudium voraus. So nehmen wir Begriffe wie Laufzeit eines Algorithmus, Turingmaschine, O-Notation oder einfache graphentheoretische Notationen als bekannt an. Informationen zu diesen Themengebieten, welche eng mit den hier behandelten Sachverhalten in Verbindung stehen, finden sich z.B. in den Lehrbüchern [9, 18, 23].

1 Ein einführendes Beispiel: Vertex Cover

Als einführendes Beispiel stellen wir ein aus der Graphentheorie bekanntes Problem vor, das sog. *Vertex Cover* (deutsch: Knotenüberdeckungsproblem). Dieses Problem ist auch unter dem Namen *Node Cover* bekannt. Wir werden im Verlauf der Vorlesung mehrfach auf dieses Beispiel zurückgreifen.

Beispiel 1.1. Gegeben: Ein ungerichteter Graph $G = (V, E)$, wobei wie üblich V die Knotenmenge und E die Kantenmenge bezeichnet, und eine natürliche Zahl k.
Frage: Gibt es eine Teilmenge $C \subseteq V$ der Mächtigkeit $|C| \leq k$ derart, daß jede Kante aus E mindestens einen Endpunkt in C hat?

In Abbildung 1 sind zwei Vertex Cover der Größe $k = 4$ für einen vorgegebenen ungerichteten Graphen dargestellt. Man erkennt, daß eine solche Knotenüberdeckung zu festem k im allgemeinen nicht eindeutig zu sein braucht. Für den gegebenen Graphen überprüft man leicht, daß kein Vertex Cover der Größe 3 existiert.

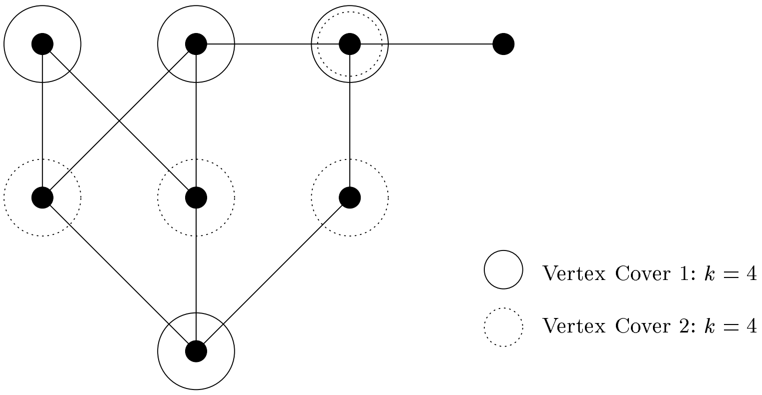


Abbildung 1: Vertex Cover für einen ungerichteten Graphen

Wir halten einige wichtige Anmerkungen zu Vertex Cover fest:

- Vertex Cover ist *NP-vollständig*. Historisch gesehen, ist Vertex Cover sogar eines der allerersten als *NP*-vollständig nachgewiesenen Probleme (vgl. [15]).
- Vertex Cover ist relevant in der Praxis, z.B. in der algorithmischen Biologie zur Behandlung von sog. Konfliktgraphen[1].

[1] Siehe hierzu etwa das DARWIN-Projekt der ETH Zürich unter `http://cbrg.inf.ethz.ch/VertexCover.html`

- Vielfach wird Vertex Cover also mit *heuristischen Verfahren* gelöst.

Wenden wir uns nun der algorithmischen Seite des Problems zu. Zunächst stellen wir zwei verschiedene Polynomzeit-*Approximationsalgorithmen* für Vertex Cover vor.

1.Versuch: „Greedy“-Heuristik:

> starte mit $C := \emptyset$;
> **while** (es gibt noch Kanten in G) **do**
>   Nimm den Knoten mit größter Anzahl von Nachbarn,
>   füge ihn zu C hinzu und lösche ihn zusammen
>   mit seinen anliegenden Kanten aus G

Hierbei handelt es sich um keinen guten Approximationsalgorithmus:

Aufgabe 1.2. Konstruiere einen Graphen, der unter obigem Algorithmus eine Lösung C besitzt, welche $\ln n$ mal so groß ist, wie das Optimum (n = Anzahl der Knoten).

2.Versuch: "Trivialer Ansatz":

> starte mit $C := \emptyset$;
> **while** (es gibt noch Kanten in G) **do**
>   Nimm irgendeine Kante $\{u, v\}$,
>   füge sowohl u als auch v zu C hinzu und
>   lösche beide aus G zusammen mit den anliegenden Kanten

Lemma 1.3. *Obiger Algorithmus („Trivialer Ansatz“) liefert ein Vertex Cover, welches schlimmstenfalls doppelt so groß ist wie das optimale Vertex Cover. (kurz: Obiger Algorithmus liefert eine Faktor-2-Approximation.)*

Beweis.
Beobachtung: Nach Ablauf des Algorithmus enthält C genau $\frac{1}{2}|C|$ Kanten aus G, von denen keine zwei einen gemeinsamen Knoten haben.

Es gilt: *Jedes* Vertex Cover, insbesondere auch das optimale, muß mindestens einen Knoten von jeder dieser Kanten beinhalten.

Mit obiger Beobachtung muß das Optimum also mindestens $\frac{1}{2}|C|$ Knoten beinhalten, also liefert der vorgestellte Algorithmus eine Faktor-2-Approximation. $\square$

Bemerkung 1.4.
- Obiger Algorithmus ist im wesentlichen der beste bekannte Approximationsalgorithmus für Vertex Cover. Es geht lediglich asymptotisch besser: Nach einem Ergebnis von Monien/Speckenmeyer [19] und Bar-Yehuda/Even [4] existiert eine Approximation mit Faktor $\left(2 - \frac{\log \log n}{2 \log n}\right)$.

- Nach einem Ergebnis von Håstad [16] kann es keinen besseren Approximationsalgorithmus als einen solchen mit Faktor 1.1666, es sei denn $P = NP$.

Zuletzt geben wir einen einfachen, *parametrisierten Algorithmus* für Vertex Cover an:

Tiefenbeschränkter Suchbaum:

1. Konstruiere einen vollständigen Binärbaum der Tiefe k.
2. Markiere den Wurzelknoten mit $(G, \emptyset)$.
3. Die restlichen Baumknoten werden *rekursiv* wie folgt markiert:
   Sei (H, S) ein markierter Baumknoten mit zwei unmarkierten Kindern. Wähle irgendeine Kante $\{u, v\}$ aus der Kantenmenge von H.
   (a) Markiere das linke Kind mit $(H - u, S \cup \{u\})$.[2]
   (b) Markiere das rechte Kind mit $(H - v, S \cup \{v\})$.
4. **if** Es gibt einen Baumknoten mit Markierung $(\substack{\text{leerer}\\\text{Graph}}, S')$
   then S' ist ein Vertex Cover der Größe $\leq k$
   else Es existiert kein Vertex Cover der Größe $\leq k$ in G.

Die Korrektheit des Algorithmus ist sofort einsichtig. Beachte hierzu, daß — wie wir dies schon bei unserem 2. Approximationsalgorithmus festgestellt hatten — von *jeder* Kante in G mindestens ein Endpunkt im Vertex Cover liegen muß.

Dieser Algorithmus hat die Laufzeit

$$O(2^k \cdot |G|),$$

wo 2^k der Baumgröße entspricht und pro Baumknoten der Graph im wesentlichen einmal durchlaufen werden muß. Der Faktor $|G|$ bezeichne dabei die Graphgröße.

Zentrale Beobachtung:
Ist k „klein“ im Vergleich zur Eingabegröße n, so ist obiger Algorithmus „effizient“.

Bemerkung 1.5.
- Oftmals ist die Annahme eines im Vergleich zur Eingabegröße n kleinen Parameters k sehr natürlich, so z.B. bei der Vertex Cover Anwendung in der algorithmischen Biologie.
- Für $k = O(\log n)$ liefert obige Suchbaummethode einen Polynomzeitalgorithmus, ist also Vertex Cover in der Komplexitätsklasse P.

[2] Unter der Kurzschreibweise $H - u$ verstehen wir das Löschen des Knoten u und aller anliegenden Kanten aus H.

2 Standortbestimmung

In vielen Anwendungsfällen hat es den Anschein, die Welt sei voll von „hartnäckigen Problemen“ (*NP*-harten Problemen). Ungeachtet ihrer Schwierigkeit müssen diese Probleme in der Praxis jedoch gelöst werden. Als Beispiel hatten wir bereits das Problem Vertex Cover kennengelernt, für welches es wegen seiner *NP*-Vollständigkeit im allgemeinen (worst case) wohl keinen effizienten Algorithmus geben kann.

Wir wollen in diesem Abschnitt der Frage nachgehen, welche Auswege man finden kann, um das oben angeführte Dilemma zu umgehen.

2.1 Ansätze zur Behandlung komplexitätstheoretisch harter Probleme

In diesem Abschnitt stellen wir die gegenwärtig bekannten Ansätze zur Behandlung kombinatorisch schwieriger Probleme vor und geben eine kurze, vergleichende Bewertung.

2.1.1 Average Case- statt Worst Case-Analyse

Cantus firmus: „In der Praxis interessieren oft die durchschnittlichen Laufzeiten mehr als die schlimmst möglich auftretenden.“

Die Laufzeit eines Algorithmus soll also in irgendeiner Form über alle Eingabeveteilungen „gemittelt“ werden.

Probleme:

- Was ist eine realistische Eingabeverteilung?
- Selbst für einfachste Algorithmen und Verteilungen ist die Average Case-Analyse in der Regel mathematisch sehr schwierig. In der Literatur sind — mit einigen wenigen Ausnahmen — für einen solchen Ansatz recht wenig Ergebnisse und Methoden bekannt.
- Ein Problem kann auch im Average Case hart bleiben...

2.1.2 Approximation

Cantus firmus: „Sei weniger anspruchsvoll und sei auch mit nicht-optimalen Lösungen zufrieden.“

Dieser Ansatz gilt als der (zumindest von den Theoretikern) am meisten favorisierte Zugang. Dies spiegelt sich in einer sehr großen Anzahl von Veröffentlichungen wider.

Probleme:

- Viele der Probleme erweisen sich als hart zu approximieren: Im Fall von Vertex Cover hat der beste bekannte Approximationsalgorithmus Faktor 2 (vgl. Kapitel 1). Es gibt aber auch Probleme, die nur weitaus schlechtere Faktoren zulassen. Als Beispiel sei *Clique* genannt. Hier ist ein Approximationsalgorithmus mit Faktor $O(\frac{|V|}{(\log |V|)^2})$ bekannt, dabei ist V die Knotenmenge des Eingabegraphen. Als untere Schranke wurde bereits nachgewiesen, daß kein besserer Approximations-Faktor als $|V|^{\frac{1}{4}-\varepsilon}$ für jedes $\varepsilon > 0$ gefunden werden kann (vgl. [11]).
- Der Ansatz macht nur Sinn bei Optimierungsproblemen, nicht aber bei „reinen“ Entscheidungsproblemen wie das Erfüllbarkeitsproblem aussagenlogischer Formeln (Sat), etc.

2.1.3 Randomisierte Algorithmen

Cantus firmus: „Vertraue auf den Zufall, sei auch mit Ergebnissen zufrieden, die nur mit hoher Wahrscheinlichkeit korrekt sind oder nur mit hoher Wahrscheinlichkeit in kurzer Laufzeit zu erzielen sind.“

Dieser Ansatz findet in vielen Bereichen Anwendungen: Kryptographie (Primzahltests), algorithmische Geometrie, Pattern Matching (Mustererkennung), etc.

Randomisierte Algorithmen[3] sind oftmals sehr einfach zu implementieren, meist gar deutlich einfacher als die entsprechenden deterministischen Algorithmen. Außerdem ist bereits eine Vielzahl von Techniken und Methoden gut bekannt.

Probleme:

- „Wahrscheinlich“ taugt die Randomisierung nicht, um NP-harte Probleme in Polynomzeit zu lösen. (Für Eingeweihte: Komplexitätstheoretisch vermutet man $P = BPP$.)
- Viele randomisierte Algorithmen lassen sich zu deterministischen Algorithmen „derandomisieren“.

[3] Man vergleiche hierzu das Vorlesungsskript zur gleichnamigen Vorlesung. Dieses kann bezogen werden unter: `http://www-fs.informatik.uni-tuebingen.de/~niedermr`

2.1.4 Heuristische Verfahren

Cantus firmus: „Die in der Praxis auftretenden Probleme sind meist nicht so hart wie sie sein könnten und in der Regel klappt's, diese effizient zu lösen — aber wir garantieren für nichts.“

Unter diesem Ansatz, den man als *die* Methode der Praktiker bezeichnen darf, verstehen wir Verfahren wie etwa: Simulated Annealing, Genetische Algorithmen, Neuronale Netze, Künstliche Intelligenz, Tabu Search, Branch & Bound, etc.

Problem: In der Regel lassen sich keine „stichhaltigen“, mathematisch beweisbaren Aussagen über die Leistung solcher Algorithmen treffen.

Viele parametrisierte Algorithmen taugen auch als Heuristiken oder sind zumindest als Subroutinen von heuristischen Verfahren einsetzbar. Es ist denkbar, daß sich über parametrisierte Algorithmen sogar ein systematischer Zugang zur Entwicklung von heuristischen Verfahren ergibt. Dies ist ein neuer Forschungstrend.

2.1.5 Neue Berechnungsmodelle

DNA-Computer: „Rechnen mit molekularer 'brute force' “.
Quantum-Computer: „Parallelisierung dank quantenmechanischer Effekte.“

Probleme:

- Praktische Implementierung noch in weiter Ferne.
- Es ist unklar, ob solche Ansätze allgemein anwendbar, oder nur auf spezielle Probleme zugeschnitten sind.

2.1.6 Parametrisierte Algorithmen

Cantus firmus: „Nicht alle Formen von 'Härte' sind gleich geschaffen.“

Idee: Beschränke die bei NP-harten Problemen scheinbar inhärente kombinatorische Explosion auf den Parameter. Der $O(2^k \cdot n)$-Algorithmus für Vertex Cover zeigt, daß die exponentielle Explosion bezüglich dem Parameter k und nicht bezüglich der Eingabegröße n auftritt. Man kann sich diesen Sachverhalt durch das Bild in Abbildung 2 versinnbildlichen.

Probleme werden wir im Rahmen der Vorlesung kennenlernen....

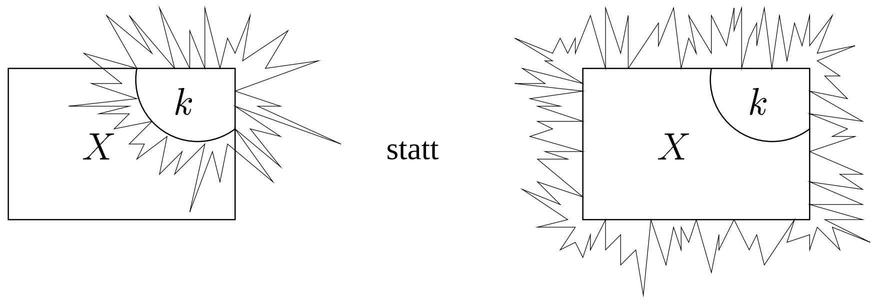


Abbildung 2: Kombinatorische Explosion NP-harter Probleme auf Parameter k beschränkt statt auf ganze Eingabegröße $n = |X|$ einwirkend.

2.2 Aspekte der Parametrisierung

In diesem Abschnitt lernen wir einige grundlegende Prinzipien und Beobachtungen kennen, die die Betrachtung parametrisierter Komplexität motivieren.

2.2.1 Laufzeit-Argumente

Wir geben einen kurzen Vergleich zwischen exponentiellen und polynomiellen Laufzeiten. Unsere Angaben stützen sich auf Annahme einer Eingabegröße $n = 50$ und einen Rechner, welcher mit 10^8 Operationen pro Sekunde arbeitet.

Polynomiell		Exponentiell	
Komplexität	Laufzeit	Komplexität	Laufzeit
n^2	25 μsec	1.5^n	6 min
n^3	1 min	2^n	130 Tage
n^5	3 sec	3^n	$230 \cdot 10^6$ Jahre

Leider sind für NP-harte Probleme bestenfalls Algorithmen mit exponentieller Laufzeit bekannt.

Im folgenden werden wir 3 parametrisierte Probleme (aus der Graphentheorie) kennenlernen und uns die Frage stellen, ob diese „effiziente“ parametrisierte Algorithmen zulassen.

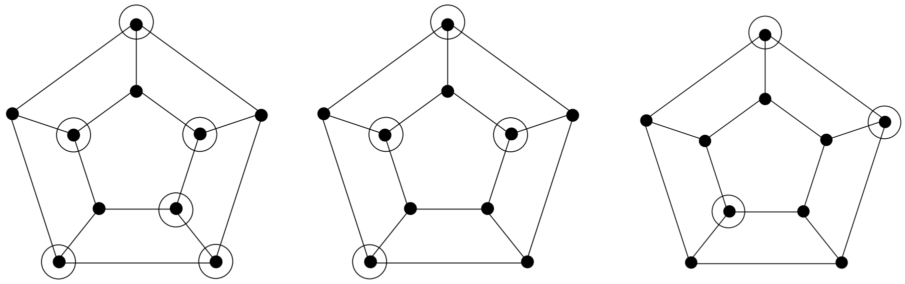
Gegeben sei stets ein Graph $G = (V, E)$ und ein Parameter $k \in \mathbb{N}$.

Problem 1: Vertex Cover
Gibt es ein $S \subseteq V$ mit $|S| \leq k$, so daß jede Kante aus E mindestens einen Endpunkt in S besitzt.

Problem 2: Independent Set
Gibt es ein $S \subseteq V$ mit $|S| \geq k$, so daß keine 2 Knoten in S in G benachbart sind.

Problem 3: Dominating Set
Gibt es ein $S \subseteq V$ mit $|S| \leq k$, so daß jeder Knoten in V mindestens einen Nachbarn in S besitzt.

Beispiele: siehe Abbildung 3.



Vertex Cover $k = 6$. Independent Set $k = 4$. Dominating Set $k = 3$.

Abbildung 3: Beispiele für Vertex Cover, Independent Set und Dominating Set. Möglichkeiten für die entsprechend gesuchten Mengen sind jeweils eingekreist.

Der „Brute-Force“-Ansatz zur Lösung aller 3 Probleme:
„Probiere *alle* Teilmengen S der Größe k durch.“

Da es genau $\binom{n}{k}$ solche Teilmengen gibt (wobei $n = |V|$), können wir mit geeigneten Datenstrukturen auf diese Weise einen Algorithmus mit Laufzeit $O(n^{k+1})$ angeben. Hier wurde $\binom{n}{k} \approx O(n^k)$ verwendet.

Wie wir gesehen haben (siehe Kapitel 1), können wir für Vertex Cover auch einen $O(2^k \cdot n)$-Algorithmus angeben.

Wir vergleichen die Laufzeiten des $O(n^{k+1})$-“Brute Force“-Algorithmus mit dem $O(2^k \cdot n)$-Algorithmus für Vertex Cover, indem wir uns für verschiedene Werte von n und k den Quotienten $\frac{n^{k+1}}{2^k \cdot n}$ ansehen:

	$n = 50$	n=150
k=2	625	5625
k=5	390625	31640625
k=20	$1.8 \cdot 10^{26}$	$2.1 \cdot 10^{35}$

Bemerkung 2.1. Im Gegensatz zu Vertex Cover sind für Independent Set und Dominating Set im wesentlichen keine besseren Ansätze als oben genannte „Brute-Force"-Verfahren bekannt.

Fazit 2.2. Nicht alle „parametrisierten Probleme" besitzen effiziente parametrisierte Algorithmen.

Wir suchen also nach Algorithmen, die möglichst kleine, *ausschließlich* auf k beschränkte exponentielle Laufzeitkomponente haben.

Für Vertex Cover klappt dies — wie bereits gesehen — sehr gut: Uns ist bereits ein $O(2^k \cdot n)$-Algorithmus bekannt. Die Laufzeit läßt sich für dieses Problem sogar auf $O(1.292^k \cdot n)$ reduzieren [22][4]. Abschließend ein kleiner Vergleich von 2^k und 1.292^k, durch Berechnung des Quotienten $\frac{2^k}{1.292^k} \approx 1.548^k$ für verschiedene Werte von k:

k=2	≈ 9
k=10	≈ 80
k=20	≈ 6240
k=40	$\approx 3.9 \cdot 10^7$
k=80	$\approx 1.5 \cdot 10^{15}$

Fazit 2.3. Selbst vermeintlich kleine Änderungen in der exponentiellen Basis eines parametrisierten Algorithmus können einen riesigen Effizienzgewinn bewirken.

2.2.2 Parametrisierte Algorithmen — eine Normalität?

Parameter (beschränkter Größe) treten in natürlicher Weise in vielen Anwendungsgebieten auf, z.B.:

VLSI-Entwurf: z.B. entspricht bei Problemen, die beim Layout in der Chip-Produktion auftreten, k etwa der beschränkten Anzahl von Verdrahtungsschichten. Typische Werte hierfür sind $k \leq 30$.

Netzwerk-Probleme: z.B. optimale Positionierung einer „kleinen" Anzahl von Einrichtungen in einem großen Netzwerk.

[4] Jüngst weiter verbessert zu $O(1.271^k \cdot n)$ [8].

Logik- und Datenbank-Probleme: z.B. Eingabe, bestehend aus einer Formel (klein) und einer anderen, großen Struktur (etwa einer Datenbank).

Robotik: z.B. beschränkte Anzahl von Freiheitsgraden (typischerweise $k \leq 10$) bei komplexen Bewegungs-/Routen-Planungen.

Künstliche Intelligenz: z.B. Planungsprobleme.

Algorithmische Biologie: z.B. Untersuchung/Vergleich von DNA-Sequenzen der Länge n für k Spezien.

Oftmals sind die Parameter auch in den tatsächlich auftretenden Problemen „versteckt" und auf den ersten Blick nicht offenkundig. Als Beispiel hierfür sollen uns wieder einige Probleme der Graphentheorie dienen. Es folgt ein kleiner Exkurs zu speziellen „Weite-Metriken" für Graphen.

Definition 2.4. Sei $G = (V, E)$ ein Graph.

(i) Eine *Baumzerlegung* von G ist ein Paar $< \{X_i \mid i \in I\}, T >$, wobei $X_i \subseteq V$ und T ein Baum mit Knotenmenge aus I ist, so daß gilt:

- $\bigcup_{i \in I} X_i = V$,
- $\forall$ Kanten $\{u, v\} \in E \; \exists \, i \in I : \;\; u \in X_i$ und $v \in X_i$,
- $\forall i, j, k \in I$: Falls j auf dem Pfad zwischen i und k in T liegt, so gilt $X_i \cap X_k \subseteq X_j$.

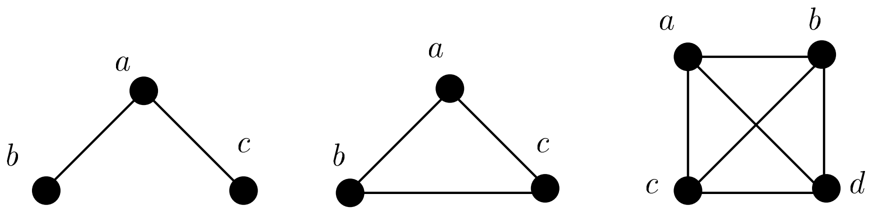
Die *Weite* einer Baumzerlegung $< \{X_i \mid i \in I\}, T >$ ist definiert als $\max_{i \in I}\{|X_i| - 1\}$.

(ii) Die *Baumweite* $w(G)$ von G ist der minimale Wert k, so daß G eine Baumzerlegung der Weite k besitzt.

Beispiel 2.5. Beispiele für die Bestimmung von Baumweiten finden sich in den Abbildungen 4 und 5.

Aufgabe 2.6. Es sei $G = (V, E)$ ein Graph.

1. Zeige, daß für eine Clique C der Größe k stets gilt: $w(C) = k - 1$.
2. Sei $< \{X_i \mid i \in I\}, T >$ eine Baumzerlegung von G, dann zeige:



Graph G_1 Graph G_2 Graph G_3

Baumzerlegung für G_1:

$\{a, b\}$ — $\{a, c\}$

$w(G_1) = 1$

Versuch für G_2:

$\{a, b\}$ mit den Nachbarn $\{b, c\}$ und $\{a, c\}$

aber: $\{b, c\} \cap \{a, c\} \not\subseteq \{a, b\}$

also: Baumzerlegung für G_2:

$\{a, b, c\}$

$w(G_2) = 2$

Baumzerlegung für G_3:

$\{a, b, c, d\}$

$w(G_3) = 3$

Abbildung 4: Baumzerlegungen für einfache Graphen

(a) Für $u \in V$ bildet die Menge aller Baumknoten $i \in I$ mit der Eigenschaft $u \in X_i$ einen Teilbaum von T.

(b) Es sei C eine Clique in G. Dann existiert ein $i \in I$ mit $C \subseteq X_i$.

3. Mit (1) und (2) zeige: Für die Baumweite $w(G)$ von G gilt die Abschätzung:

$$\min\Big\{|C| - 1 : C \text{ ist eine Clique in } G\Big\} \;\leq\; w(G) \;\leq\; |V|.$$

Eine Reihe von Anwendungsbeispielen bringt einen versteckten Parameter k in Form einer implizit gegebenen, beschränkten Baumweite mit sich:

- Es wurde gezeigt [28], daß für „strukturierte imperative Programme“ der Kontrollflußgraph Baumweite ≤ 10 hat. Damit ist das Register-Zuordnungsproblem für solche Graphen (kleines k) mit parametrisierten Algorithmen lösbar.

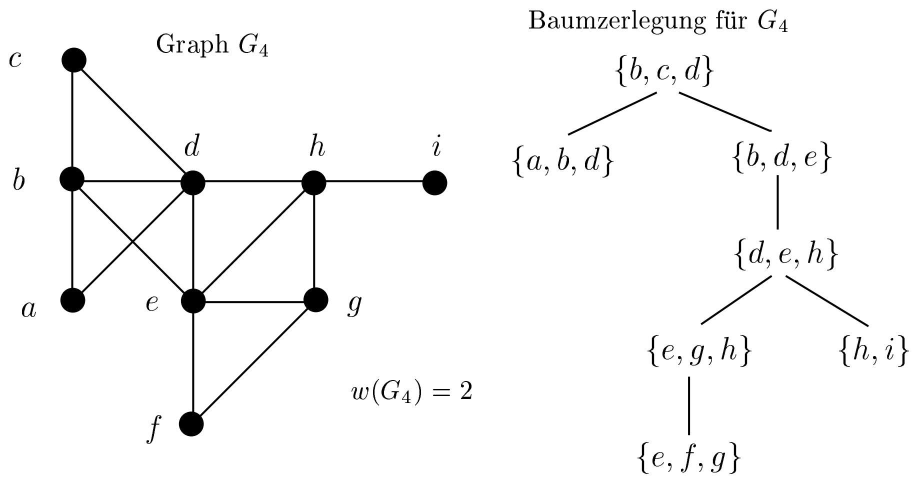


Abbildung 5: Baumzerlegung für einen komplizierteren Graphen

- Es wurde vorgeschlagen, daß die syntaktische Struktur von natürlichen Sprachen mit Abhängigkeitsgraphen der Pfadweite < 10 modelliert werden kann. (Die Pfadweite ist ein Spezialfall der Baumweite.)
- In der Logikprogrammierung haben die logischen Formeln, welche in natürlichen Programmen vorkommen, eine kleine, beschränkte Tiefe von Let-Anweisungen. Damit ergibt sich eine Vereinfachung der Typeninferenz, obwohl es sich hier i.allg. um ein vollständiges Problem für deterministische Exponentialzeit handelt.
- Abschließend zur Motivation noch ein „Meta-Argument“: In der Praxis betrachtete Probleme kommen von Menschen, und die Beschränkung des menschlichen Geistes impliziert auch oft kleine Parameter, die in den Problemen wieder auftauchen...

3 Grundlegende Definitionen

Wir erinnern zunächst an einige grundlegende Definitionen aus der Komplexitätstheorie.

P: Klasse von Entscheidungsproblemen, die in polynomieller Laufzeit von einer *deterministischen* Turingmaschine erkannt werden können.

Beispiel 3.1. Beispiele für Probleme, welche deterministisch in polynomieller Zeit gelöst werden können:

- Sortieren: $O(n \log n)$.
- Erkennung kontextfreier Sprachen mittels des CYK-Algorithmus: $O(n^3)$.
- Matrixmultiplikation für $n \times n$-Matrizen mittels Standardmethode: $O(n^3)$.

NP: Klasse von Entscheidungsproblemen, die in polynomieller Laufzeit von einer *nichtdeterministischen* Turingmaschine erkannt werden können.

Beispiel 3.2. Beispiele für Probleme, welche nichtdeterministisch in polynomieller Zeit gelöst werden können:

- Alle Probleme aus P, da P $\subseteq NP$.
- Erfüllbarkeitsproblem für aussagenlogische Formeln in konjunktiver Normalform, kurz Sat.
- Vertex Cover.
- Dominating Set.

NP-vollständige Probleme: Diejenigen Probleme in NP, für die gilt, daß jedes andere Problem in NP auf sie mittels eines deterministischen Polynomzeitalgorithmus reduziert werden kann:

Sei A ein NP-vollständiges Problem und L ein beliebiges anderes Problem aus NP. Dann gilt:

$$L \leq_p A,$$

d.h. es existiert eine in deterministischer Polynomzeit berechenbare Funktion $f : \Sigma^* \to \Sigma^*$, so daß für alle $x \in \Sigma^*$ gilt:

$$x \in L \quad \Leftrightarrow \quad f(x) \in A.$$

Dabei ist Σ ein Eingabealphabet und f wird die *Reduktionsfunktion* genannt.

Beispiel 3.3. Es gilt: Vertex Cover $\leq_p$ Independent Set (und umgekehrt). Die Reduktionsfunktion ist im wesentlichen gegeben durch $f((G,k)) := (G, n-k)$, wo G den Graphen $G = (V, E)$ bezeichne mit $n = |V|$.

NP-harte Probleme: Wie *NP*-vollständige Probleme, nur daß *nicht* gefordert wird, daß das Problem selbst in *NP* liegen muß.

Beispiel 3.4.

- *NP*-vollständig: Sat, Vertex Cover, Dominating Set,...
- *NP*-hart: Sat, Vertex Cover, Dominating Set, ...
  aber auch „härtere" Probleme wie QBF, Quantified Boolean Formula (Gegeben: Eine „quantifizierte boolesche Formel", d.i. eine aussagenlogische Formel, in welcher auch der Allquantor „$\forall$" und der Existenzquantor „$\exists$" vorkommen können. Frage: Ist diese quantifizierte Boolesche Formel „wahr"?) QBF ist PSPACE-vollständig, und deshalb wohl „härter" als *NP*-vollständige Probleme.

Wir kehren nun zu den parametrisierten Problemen zurück. Sei im folgenden Σ ein Eingabealphabet konstanter Größe.

Definition 3.5. Ein *parametrisiertes Problem* ist eine Sprache $L \subseteq \Sigma^* \times \Sigma^*$. Falls $< x, k > \in L$, so nennen wir k den *Parameter*.

Üblicherweise betrachten wir parametrisierte Probleme speziell als Untermenge von $\Sigma^* \times \mathbb{N}$.

Beispiel 3.6. Wir geben eine Reihe von Beispielen für derartige Problemstellungen.

- **Vertex Cover**, **Independent Set**, **Dominating Set** (zu deren Definition siehe Abschnitt 2.2.1).
- **Clique:**

  Gegeben: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$.

  Frage: Existiert $V' \subseteq V$ mit $|V'| \geq k$, so daß V' einen vollständigen Teilgraphen in G bildet, d.h. jeder Knoten in V' ist mit jedem anderen Knoten in V' verbunden.

- **MaxSat:**

  Gegeben: Eine aussagenlogische Formel F in konjunktiver Normalform und $k \in \mathbb{N}$.

Frage: Existiert eine Belegung, so daß mindestens $\lceil \frac{m}{2} \rceil + k$ Klauseln wahr sind, wobei m die Anzahl aller Klauseln in F ist? (Bemerkung: Es ist trivial, eine Belegung zu finden, die mindestens $\lceil \frac{m}{2} \rceil$ Klauseln erfüllt: Wähle hierzu eine beliebige, zufällige Belegung, dann erfüllt entweder diese, oder ihre Komplementbelegung[5] die genannte Voraussetzung.)

- **Weighted qCNF Sat:**

  Gegeben: Eine aussagenlogische Formel F in konjunktiver Normalform mit höchstens q Literalen per Klausel und $k \in \mathbb{N}$.

  Frage: Hat F eine erfüllende Belegung vom Gewicht k, d.h., eine erfüllende Belegung, bei der genau k Variablen mit „wahr" belegt werden?

- **Short NTM computation:**

  Gegeben: Eine nichtdeterministische Turingmaschine M, ein Eingabewort x für M und $k \in \mathbb{N}$.

  Frage: Kann M bei Eingabe x eine Endkonfiguration nach höchstens k Schritten erreichen?

Die zentrale Frage, die wir uns stellen, lautet:
Welche dieser Probleme können für „kleines" k „effizient" gelöst werden?

Definition 3.7. Ein parametrisiertes Problem L heißt *fixed parameter tractable*, wenn in Laufzeit $f(k)n^c$ festgestellt werden kann, ob $(x, k) \in L$. Dabei seien $n := |x|$, c eine Konstante unabhängig von sowohl n als auch k, und $f : \mathbb{N} \to \mathbb{R}$ eine beliebige Funktion.

Die Familie aller fixed parameter tractable parametrisierten Probleme bezeichnen wir mit FPT.

Wie bereits gezeigt, ist Vertex Cover in FPT.

Bemerkung 3.8.

- Downey und Fellows unterscheiden in ihrem Buch die Begriffe *uniform*, *streng uniform* und *nicht-uniform fixed parameter tractable*, je nachdem ob man z.B. von der Funktion f Berechenbarkeit verlangt, bzw. ob für jedes k ein anderer Algorithmus zugelassen werden darf. Wir sind hier nur an dem einfachsten Fall interessiert, d.h. *ein* Algorithmus für k und f in der Regel berechenbar.

[5]Darunter versteht man jene Belegung, die gegenüber der ursprünglichen Belegung die Variablenwerte 0 und 1 vertauscht.

- Die Praxisrelevanz von FPT steht und fällt mit der Güte der gefundenen Funktion f; aus konzeptionellen Gründen wird i.a. aber f als beliebig wachsende Funktion zugelassen, also wäre z.B. auch

$$f(k) = 2^{2^{2^{2^{2^k}}}}$$

(oder noch „schlimmeres" Wachstum) erlaubt.

4 Grundlegende Methoden

Wir lernen nun ein paar Grundtechniken zur Entwicklung parametrisierter Algorithmen kennen.

4.1 Reduktion auf den Problemkern

Grundidee: Reduziere gegebene Probleminstanz $\mathcal{I}$ auf eine „äquivalente“ Instanz $\mathcal{I}'$, so daß die Größe von $\mathcal{I}'$ allein durch eine Funktion in Abhängigkeit von dem Parameter k beschränkt werden kann. (Danach kann dann in $\mathcal{I}'$ etwa die Methode der „erschöpfenden Suche“ angewandt werden.)

Beispiel 1: Vertex Cover

Der Algorithmus von Buss

Eingabe: Ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$.
Ausgabe: Vertex Cover (VC) der Größe $\leq k$, falls ein solches existiert.

1. $H :=$ {Knoten in G mit mehr als k Nachbarn};
   if ($|H| > k$) **then**
   $\nexists$ VC der Größe $\leq k$; **exit**
   else begin
   Lösche alle Knoten in H
   (zusammen mit ihren inzidenten Kanten) aus G;
   $k' := k - |H|$;
   Lösche alle isolierten Knoten, d.h. Knoten ohne Nachbarn;
   end;
2. **if** (resultierender Graph hat mehr als $k \cdot k'$ Kanten) **then**
   $\nexists$ VC der Größe $\leq k$; **exit**;
3. Durchsuche den verbleibenden Graphen nach einem VC der Größe k', z.B. mit Hilfe des in Kapitel 1 beschriebenen, tiefenbeschränkten Suchbaums.

Die zentrale Idee zum Beweis der **Korrektheit** ist die Feststellung, daß ein Knoten vom Grad $> k$ stets ein Teil des VC der Größe $\leq k$ sein *muß*. In Schritt 1 werden all diese Knoten ins VC aufgenommen und es verbleiben k' Knoten, die noch zum gesuchten VC hinzugenommen werden dürfen. Diese können im verbleibenden Graphen in Schritt 2 allerdings maximal

Grad k besitzen. Es können mit den verbleibenden k' Knoten also maximal $k \cdot k'$ Kanten abgedeckt werden. Wird Schritt 3 erreicht, so liegt ein Graph zugrunde, welcher maximal $k \cdot k' \leq k^2$ Kanten besitzt.

Hieraus ergibt sich auch die **Zeitkomplexität**

$$O(kn + 2^k k^2),$$

wobei $n := |V|$. Der erste Term entspricht dabei der Zeitkomplexität der Schritte 1 und 2. Bei der Wahl geeigneter Datenstrukturen können diese in $O(kn)$ Schritten ausgeführt werden. Der zweite Term entspricht der Komplexität unseres Suchbaum-Algorithmus aus Kapitel 1 zur Suche eines VC der Größe $k' \leq k$, angewandt auf den verbleibenden Restgraphen mit maximal k^2 Kanten.

Wir fassen zusammen:

Satz 4.1. *Vertex Cover kann in Zeit $O(kn + 2^k k^2)$ gelöst werden.*

Bemerkung 4.2.

- Entscheidende Verbesserungen in der Zeitkomplexität von Vertex Cover wurden durch die Verkleinerung der Suchbaumgröße (in obigem Schritt 3) erreicht (vgl. hierzu [3, 8, 13, 22, 27]).
- Obwohl die Reduktion auf den Problemkern üblicherweise S. Buss [6] zugeschrieben wird, finden sich im wesentlichen gleiche Ideen schon in früheren Arbeiten aus dem Bereich des VLSI-Designs, z.B. [14].

Beispiel 2: Hitting Set für Mengen der Größe 3 (3HS)

Das Problem *3HS* ist wie folgt gegeben.

Gegeben: Eine Menge C von dreielementigen Teilmengen einer Menge S und ein $k \in \mathbb{N}$.
Frage: Enthält S eine „Hitting Set“ für C der Größe höchstens k, d.h. $\exists\, S' \subseteq S, |S'| \leq k$, so daß S' mindestens ein Element aus jeder der Mengen in C enthält?

Beachte, daß 2HS genau dem uns bekannten Vertex Cover (VC) entspricht, in diesem Sinn also 3HS eine Erweiterung von VC ist, wo wir uns die „Kanten“ als Verbindung von jeweils 3 „Knoten“ vorstellen können (Stichwort „Hypergraph“). Wie VC ist auch 3HS *NP*-vollständig (vgl. [15]).

Soll es ein S' der Größe höchstens k geben, so gelten folgende Beobachtungen:

1. Für festes $x, y \in S$ gibt es höchstens k Mengen der „Form" $\{x, y, *\}$ (wo $*$ ein beliebiges Element bezeichnet). In der Tat, gäbe es mehr als k Möglichkeiten für $*$, so *muß* x oder y in S' sein und damit könnten alle Mengen der Form $\{x, y, *\}$ aus C entfernt werden.

2. Für festes $x \in S$ gibt es höchstens k^2 Mengen der Form $\{x, *, *\}$, denn: Dank Punkt 1. wissen wir, daß x zusammen mit einem anderen Element y in höchstens k Mengen vorkommen kann. Gäbe es mehr als k^2 Mengen, die x beinhalten, so müßte wegen $|S'| \leq k$ folglich $x \in S'$ gelten. Damit könnten dann alle Mengen der Form $\{x, *, *\}$ aus C entfernt werden.

3. Aus 2. folgt insbesondere, daß jedes Element x in höchstens k^2 dreielementigen Mengen auftreten kann. Daraus folgt wegen $|S'| \leq k$, daß C aus höchstens $k \cdot k^2 = k^3$ dreielementigen Mengen bestehen kann. Der Problemkern hat also die Größe k^3.

Wir fassen wiederum zusammen:

Lemma 4.3. *3HS besitzt einen Problemkern der Größe $O(k^3)$. Dieser kann in Zeit linear in der Eingabegröße gefunden werden.*

Beweis.
Die Größe des Problemkerns wurde mit obigen Beobachtungen bereits nachgewiesen. Es bleibt also die Zeitschranke zu zeigen: Man überlegt sich leicht, daß man (mithilfe geeigneter Datenstrukturen) leicht zählen kann, wie oft ein Element in den Mengen vorkommt und es gegebenenfalls zusammen mit seinen Mengen aus C löscht. All dies ist in Zeit linear in der Eingabegröße möglich. □

Beispiel 3: Vererbbare Grapheneigenschaften

Wir lernen hier eine etwas allgemeiner gehaltene Anwendung der Reduktion auf den Problemkern in der Graphentheorie kennen. Vorab hierzu einige Definitionen.

Definition 4.4. Sei $G = (V, E)$ ein Graph und $\emptyset \neq U \subseteq V$. Der von U *induzierte Teilgraph* von G ist derjenige Graph, dessen Knotenmenge U ist, und dessen Kantenmenge aus allen Kanten $\{u, v\} \in E$ besteht, für die sowohl $u \in U$ als auch $v \in U$.

Definition 4.5. Eine *Grapheneigenschaft* Π ist eine Menge Π von Graphen. Jeden Graph in Π nennen wir Π-*Graph.* Eine Grapheneigenschaft Π heißt *vererbbar*, falls für jeden Π-Graph gilt, daß jeder induzierte Teilgraph von G wieder ein Π-Graph ist.

Die meisten (interessanten) Grapheneigenschaften sind in der Tat vererbbar. Man mache sich dies etwa an den Eigenschaften

1. $\Pi = \{G :\ G$ ist ein „planarer Graph“, d.h. G läßt sich in der Ebene derart darstellen, daß keine Kantenüberschneidungen auftreten $\}$

2. $\Pi = \{G :\ G$ ist ein „bipartiter Graph“, d.h. seine Knotenmenge läßt sich in zwei Teilengen derart unterteilen, daß für je zwei Knoten innerhalb derselben Teilmenge keine Kantenverbindungen bestehen.$\}$

klar.

Definition 4.6. Eine Grapheneigenschaft Π hat eine *Charakterisierung mittels Ausschlußmengen* (engl.: *forbidden set characterization*), falls es eine Menge F von Graphen gibt, so daß gilt: Ein Graph ist ein Π-Graph genau dann wenn G keinen Graph in F als induzierten Teilgraph enthält. Ein Element aus F nennen wir einen *verbotenen induzierten Teilgraphen.*

Eine Charakterisierung mittels Ausschlußmengen heißt *endlich*, falls F endlich ist.

Aufgabe 4.7. Man zeige: Eine Grapheneigenschaft Π ist vererbbar genau dann wenn sie eine Charakterisierung mittels Ausschlußmengen besitzt.

Das $\Pi_{i,j,k}$-Graphenmodifikationsproblem:

Gegeben: Ein Graph $G = (V, E)$ und $i, j, k \in \mathbb{N}$.
Frage: Läßt sich G durch Löschen von höchstens i Knoten, Löschen von höchstens j Kanten und Hinzufügen von höchstens k Kanten in einen Π-Graph umwandeln?

Graphenmodifikationsprobleme tauchen vielfach in der Literatur auf. Als jeweilige Spezialfälle des obigen Problems, die im allgemeinen auch schon NP-hart sind, ergeben sich:

- Kantenlöschungsproblem:
  Lösche die minimale Anzahl von Kanten, so daß ein Π-Graph entsteht.

- Knotenlöschungsproblem:
  Lösche die minimale Anzahl von Knoten, so daß ein Π-Graph entsteht.

- Knoten- und Kantenlöschungsproblem:
  Lösche die minimale Anzahl von Kanten und Knoten, so daß ein Π-Graph entsteht.

- Kantenhinzufügungsproblem:
  Finde die minimale Anzahl von Kanten, so daß durch deren Hinzufügen ein Π-Graph entsteht.

In der allgemeinen Situation haben wir hier also ein parametrisiertes Problem vorliegen, welches von *drei* Eingabeparametern abhängt. Wir wollen zeigen, daß das $\Pi_{i,j,k}$-Graphenmodifikationsproblem in FPT liegt, daß wir also einen Algorithmus mit Laufzeit $O(f(i,j,k) \cdot n^{O(1)})$ zur Lösung des Problems finden können.

Lemma 4.8. *Sei $G = (V, E)$ ein Graph mit $n := |V|$. Dann gilt für jede vererbbare Grapheneigenschaft Π: Falls Π für G in Zeit $T(G)$ überprüft werden kann, dann kann für jeden Graphen G, welcher nicht ein Π-Graph ist, ein minimaler verbotener induzierter Teilgraph in Laufzeit $O(n \cdot T(G))$ gefunden werden.*

Beweis.
Es bezeichne A den Algorithmus, welcher auf Eingabe G `true` ausgibt, genau dann wenn G ein Π-Graph ist.

Folgender Algorithmus konstruiert die Knotenmenge W des minimalen verbotenen induzierten Teilgraphen:

> $W := \emptyset$;
> $U := V$;
> **while** $(U \neq \emptyset)$ **do**
>   Wähle beliebigen Knoten $v \in U$;
> $U := U - \{v\}$;
>   **if** $(A(G - v) =$ `true`$)$
>   **then** $W := W \cup \{v\}$
>   **else** $G := G - v$;

Es ist klar, daß der Algorithmus nach n Teilschritten terminiert, wir erhalten also in der Tat eine Laufzeit von $O(n \cdot T(G))$, wie behauptet.

Bleibt die Korrektheit des Algorithmus nachzuweisen:
Behauptung: W induziert einen minimalen verbotenen Teilgraphen G' von G.

Wird kein Knoten v aus G gelöscht, so ist $W = V$ offensichtlich die minimale Menge, welche einen solchen verbotenen Teilgraphen induziert.

Sei nun v' der zuletzt aus G gelöschte Knoten, und sei G^* der Graph unmittelbar vor dem Löschen von v', also $G' = G^* - v'$. Da v' gelöscht wird, ist G' kein Π-Graph. Es bleibt die Minimalität von G' zu zeigen, d.h. wir müssen überprüfen, ob

$$\forall\, u \in W : \; G' - u \; \text{ ist ein } \Pi\text{-Graph.}$$

Angenommen es existiert ein $x \in W$, so daß $G'-x$ kein Π-Graph ist. Sei dann G'' derjenige Graph während Ablauf des Algorithmus, wenn x betrachtet wird. Dann ist G' ein induzierter Teilgraph von G'' und damit auch $G'-x$ induzierter Teilgraph von $G''-x$. Da Π eine vererbbare Grapheneigenschaft ist, folgt daß $G''-x$ ebenfalls kein Π-Graph ist. Damit wäre x aus G'' gelöscht worden, also $x \notin W$ im Widerspruch zu unserer Annahme. $\square$

Satz 4.9. *Sei Π eine Grapheneigenschaft mit einer endlichen Charakterisierung mittels Ausschlußmenge F. Dann ist das $\Pi_{i,j,k}$-Graphenmodifikationsproblem in FPT mit einer Laufzeit von $O(N^{i+2j+2k}|G|^{N+1})$, wobei N die maximale Größe einer Knotenmenge von Graphen in der verbotenen Menge F bezeichnet.*

Man beachte, daß in obigem Satz N eine durch das Problem gegebene Konstante ist.

Beweis.
Seien G der Eingabegraph und i, j, k die Parameter. Wir wiederholen folgende zwei Schritte bis

> *entweder* ein Π-Graph konstruiert wurde
>
> *oder* i Knoten und j Kanten gelöscht und k Kanten addiert wurden, ohne einen Π-Graph bekommen zu haben:
>
> Schritt 1: Finde einen minimalen verbotenen induzierten Teilgraph H von Π in G.
>
> Schritt 2: Modifiziere G durch entweder Löschen eines Knoten oder einer Kante aus H oder Hinzufügen einer Kante zu H.

Die **Korrektheit** des Algorithmus ist einfach nachzuprüfen. Für die **Zeitkomplexität** gelten folgende Aussagen:

- In Zeit $O(n^N)$, wo $n := |V|$, kann festgestellt werden, ob G ein Π-Graph ist. Für jeden Graphen H in F müssen dafür alle $\binom{n}{|H|}$ Teilgraphen aus G der Größe $|H|$ untersucht werden. Wegen $\binom{n}{|H|} \le n^N$ für jeden dieser Teilgraphen hat der Algorithmus, welcher bestimmt, ob G ein Π-Graph ist, die maximale Laufzeit $O(|F| \cdot n^N) = O(n^N)$, wie behauptet.
- Nach Lemma 4.8 brauchen wir daher zur Berechnung des minimalen induzierten verbotenen Teilgraphen H von G die Zeit $O(n^{N+1})$.

- Nach Definition von N folgt, daß es höchstens $\binom{N}{2}$ Möglichkeiten gibt, eine Kante von H zu löschen oder eine hinzuzufügen; außerdem gibt es höchstens N Möglichkeiten, einen Knoten aus H zu löschen. Insgesamt können also höchstens $N^i \cdot \binom{N}{2}^{j+k} = O(N^{i+2j+2k})$ verschiedene Graphen durch obigen Algorithmus erzeugt werden.

Die Gesamtlaufzeit beträgt also $O(N^{i+2j+2k} \cdot n^{N+1})$. □

Bemerkung 4.10. 1. Man vergleiche die in obigem Satz angeführte Zeitkomplexität, mit jener, die durch einen trivialen „Brute-Force-Algorithmus" erzielt werden kann: $O(n^{i+2j+2k} \cdot n^N)$. Die Tatsache, daß im ersten Faktor der Term $n^{i+2j+2k}$ durch den Ausdruck $N^{i+2j+2k}$ mit konstanter Basis ersetzt werden konnte, ist unserer Reduktion auf den Problemkern zu verdanken. Statt die Modifikation auf dem gesamten Graphen G auszuüben, ist es ausreichend, diese Modifikationen auf dem minimalen induzierten Teilgraphen H in G durchzuführen. Deren Größe ist durch die Konstante N beschränkt.

2. Beachte, daß bei diesem Ansatz auf die Endlichkeit der Ausschlußmenge F nicht verzichtet werden kann. Es sei an dieser Stelle angemerkt, daß es aber auch Graphenmodifikationsprobleme gibt, die nicht durch eine *endliche* Ausschlußmenge charakterisiert werden können und dennoch in FPT liegen, etwa das sogenannte
Minimum Fill In (Chordal Graph Completion)-Problem:

    Gegeben: Ein Graph G und $k \in \mathbb{N}$.

    Frage: Lassen sich k Kanten addieren, so daß der Graph *„chordal"* ist, d.h. jeder Kreis im Graph bestehend aus mindestens 4 Knoten enthält eine Sehne.

Im allgemeinen (unabhängig von parametrisierten Algorithmen) kann gesagt werden, daß es sich wohl stets lohnt zu untersuchen, ob für einen Algorithmus die Methode der Reduktion auf den Problemkern als eine Art „Preprocessing" möglich ist.

4.2 Tiefenbeschränkte Suchbäume

Grundidee: Finde (in polynomieller Zeit) eine „kleine Teilmenge" der Eingabe derart, daß mindestens ein Element dieser Teilmenge in einer optimalen Lösung des Problems liegen muß.

Für das Lösen von Vertex Cover entspräche diese Grundidee etwa der trivialen Beobachtung, daß zu je einer Kante (diese entspricht der „kleine Teilmenge") mindestens einer der beiden Endknoten im optimalen Vertex Cover liegen muß. Ein ähnlicher Suchbaum-Algorithmus soll nun für 3HS angegeben werden:

Beispiel 1: Hitting Set für Mengen der Größe 3 (3HS)

Zur Erinnerung wiederholen wir die Definition von 3HS (vgl. Beispiel 2 in Abschnitt 4.1)

Gegeben: Eine Menge C von dreielementigen Teilmengen einer Menge S und ein $k \in \mathbb{N}$.
Frage: Enthält S eine „Hitting Set" für C der Größe höchstens k, d.h. $\exists\, S' \subseteq S, |S'| \leq k$, so daß S' mindestens ein Element aus jeder der Mengen in C enthält?

Die angesprochenen „kleinen Teilmengen" unserer Grundidee entsprechen hier allen dreielementigen Teilmengen aus S. Man beobachtet, daß jeweils mindestens ein Element einer solchen Teilmenge in der gesuchten Hitting Set liegen muß.

Daraus erhalten wir folgenden tiefenbeschränkten Suchbaum:

1. Konstruiere einen Baum der Tiefe k, so daß jeder Knoten (abgesehen von den Blättern) 3 Kinder hat.
2. Markiere den Wurzelknoten mit $(C, \emptyset)$.
3. Die restlichen Baumknoten werden *rekursiv* wie folgt markiert:
   Sei (D, S) ein markierter Baumknoten mit drei unmarkierten Kindern. Wähle irgendeine dreielementige Menge $\{a, b, c\}$ aus D:
   (a) Markiere das linke Kind mit $(D -$ „a-Mengen"$, S \cup \{a\})$.
   (b) Markiere das mittlere Kind mit $(D -$ „b-Mengen"$, S \cup \{b\})$.
   (c) Markiere das rechte Kind mit $(D -$ „c-Mengen"$, S \cup \{c\})$.

   Dabei verstehen wir unter „x-Menge" eine beliebige Menge aus D, welche x enthält.
4. **if** ($\exists$ Baumknoten mit Markierung $(\emptyset, S')$)
   then S' ist eine Hitting Set der Größe $\leq k$
   else $\nexists$ Hitting Set der Größe $\leq k$ für C.

Satz 4.11. *3HS kann in der Zeit $O(n + 3^k \cdot k^3)$ gelöst werden, wobei n die Eingabegröße bezeichnet.*

Beweis.
Die Suchbaumgröße ist offensichtlich 3^k. Für jeden Baumknoten sind Operationen durchzuführen, welche die Zeit $O(|C|)$ benötigen. Wenden wir vor dem tiefenbeschränkten Suchbaum eine Reduktion auf den Problemkern an (siehe Abschnitt 4.1, Beispiel 2), so wissen wir mit Lemma 4.3, daß der Problemkern mit $O(k^3)$ beschränkt werden kann, d.h. mit Beginn des Suchbaum-Algorithmus gilt $O(|C|) = O(k^3)$. Zusammenfassend ist also 3HS in Zeit $O(n + 3^k k^3)$ zu lösen, wobei $O(n \log n)$ die für die Reduktion auf den Problemkern benötigte Zeit ist. □

Beispiel 2: Ein verbesserter Suchbaum für Vertex Cover

Grundidee des hier beschriebenen Suchbaum-Algorithmus ist es, bezüglich dem jeweiligen Knotengrad eine Fallunterscheidung durchzuführen (vgl. hierzu den „schlechten“ Greedy-Approximationsalgorithmus aus Kapitel 1).

Für einen Knoten v bezeichne $N(v)$ die Menge seiner Nachbarn, der *Grad* eines Knotens ist also $|N(v)|$.

Durch Preprocessing mit der üblichen Reduktion auf den Problemkern können wir ohne Einschränkung annehmen, daß der gegebene Graph die Größe $O(k^2)$ besitzt (vgl. die Ausführungen vor Satz 4.1).

Für den neuen, verbesserten Suchbaum der Tiefe k erweist es sich als vorteilhaft, die Größe mittels Rekursionsformeln der Form

$$T_i = 1 + T_{i-d_1} + T_{i-d_2} + \ldots + T_{i-d_j} \qquad (1)$$

zu beschreiben, wobei j in der Regel eine kleine Konstante ist und $T_1 = 1$ gilt.

Beispiel: In unserem vorherigen Beispiel 1 für 3HS gab es nur eine Rekursionsgleichung. Sie hatte die Form $T_i = 1 + 3T_{i-1}$, und für die Suchbaumgröße ergab sich mit $i = k$ daraus $T_k = O(3^k)$ (verwende die geometrische Reihe zum Auflösen der Rekursionsformel).

Da wir nachfolgend mehrere Fälle zu betrachten haben und zu jedem Fall eine eigene Rekursionformel gehört, ist es vorteilhaft, die Rekursionsgleichungen durch sogenannte Verzweigungsvektoren abkürzend zu beschreiben.

Der zu obiger Rekursionsgleichung (1) gehörige *Verzweigungsvektor* ist durch $(d_1, d_2, \ldots, d_j)$ gegeben. Durch ihn ist die Rekursionsgleichung eindeutig festgelegt. Aus ihm läßt sich — mittels geeigneten mathematischen Verfahren — unmittelbar die zugehörige Baumgröße bestimmen. Genauer gesagt ermittelt man daraus die „Verzweigungszahl“, d.h. die Basis des exponentiellen Faktors. Tabelle 1 zeigt exemplarisch einige derartige Berechnungen.

Verzweigungs-vektor	errechnete Basis der Baumgröße	Verzweigungs-vektor	errechnete Basis der Baumgröße
(1,1)	2.0	(1,1,1)	3.0
(1,2)	1.6180	(1,1,2)	2.4142
(1,3)	1.4656	(1,1,3)	2.2056
(1,4)	1.3803	(1,1,4)	2.1069
(2,1)	1.6180	(1,2,1)	2.4142
(2,2)	1.4142	(1,2,2)	2.0
(2,3)	1.3247	(1,2,3)	1.8929
(2,4)	1.2720	(1,2,4)	1.7549

Tabelle 1: Exemplarische Berechnung von Baumgrößen bei gegebenen Verzweigungsvektoren.

Um nun einen verbesserten Suchbaum angeben zu können führen wir eine dertaillierte Fallunterscheidung durch. Ziel dabei ist es, die Unterfälle derart zu unterscheiden, daß Verzweigungsvektoren auftauchen, aus welchen sich sehr kleine Suchbaumgrößen errechnen.

Es werde im folgenden stets der Fall mit der kleinstmöglichen Nummer angewandt; ins Vertex Cover aufgenommene Knoten werden zusammen mit ihren anliegenden Kanten aus dem Graphen gelöscht.

Zum Verständnis der nachfolgende Argumentation ist es sehr hilfreich, sich die verschiedenen Fälle anhand von kleinen Skizzen der jeweils behandelten Graphen klarzumachen. Man führe dies für die angeführte Fallunterscheidung im Detail aus.

Fall 1: $\exists$ Grad-1-Knoten v mit Nachbar a.
$\rightsquigarrow$ Nimm a in das (zu konstruierende) Vertex Cover.
Verzweigungsvektor: Keine Verzweigung der Rekursion.

Fall 2: $\exists$ Grad-5-Knoten v mit Nachbarn $N(v) = \{a, b, c, d, e\}$.
$\rightsquigarrow$ Nimm entweder v oder $N(v)$ in das Vertex Cover.
Verzweigungsvektor: (1,5).

Fall 3: $\exists$ Grad-2-Knoten v mit Nachbarn $N(v) = \{a, b\}$.

Fall 3.1: $\exists$ Kante zwischen a und b.
$\rightsquigarrow$ Nimm $N(v)$ in das Vertex Cover.
Denn: Nimmt man v ins VC, so muß nach wie vor die Kante $\{a, b\}$ überdeckt werden. Dazu muß entweder noch zusätzlich a oder b ins VC genommen werden. Dann ist es aber günstiger, gleich $N(v)$ ins VC zu nehmen, welche ja schließlich auch die Kanten von v überdecken.
Verzweigungsvektor: keine Verzweigung der Rekursion.

Fall 3.2: $\nexists$ Kante zwischen a und b und $N(a) = N(b) = \{v, a_1\}$.
$\rightsquigarrow$ Nimm $\{v, a_1\}$ in das Vertex Cover.
Denn: Nimmt man etwa a ins VC, so bedarf es noch, b mit ins VC zu nehmen, in diesem Fall ist es aber günstiger, gleich $\{v, a_1\}$ aufzunehmen. Analog argumentiert man mit b.
Verzweigungsvektor: keine Verzweigung der Rekursion.

Fall 3.3: $\nexists$ Kante zwischen a und b und $|N(a) \cup N(b)| \geq 3$.
$\rightsquigarrow$ Nimm entweder $N(v)$ oder $N(a) \cup N(b)$ in das Vertex Cover.
Denn: Wird v nicht ins VC aufgenommen, so müssen $N(v) = \{a, b\}$ ins VC genommen werden. Alternativ hierzu kann v ins VC genommen werden; dann müssen — für die Überdeckung der Kanten $\{a, v\}$ und $\{b, v\}$ — die Knoten a und b allerdings nicht mehr ins VC genommen werden. Um dennoch die (weiteren) von a bzw. b ausgehenden Kanten abzudecken, ist allerdings die Aufnahme von $N(a) \cup N(b)$ unerläßlich.
Verzweigungsvektor: (2,3) (beachte, $|N(a) \cup N(b)| \geq 3$)

Fall 4: $\exists$ Grad-3-Knoten v mit Nachbarn $N(v) = \{a, b, c\}$.

Fall 4.1: $\exists$ Kante zwischen zwei Nachbarknoten, o.E. zwischen a und b.
$\rightsquigarrow$ Nimm entweder $N(v) = \{a, b, c\}$ oder $N(c)$ in das Vertex Cover.
Denn: Ist v im VC, so muß mind. einer der beiden Nachbarn von a und b drin sein. Wäre auch noch c im VC, so wäre es mind. genau so gut, $N(v)$ selbst zu nehmen.
Verzweigungsvektor: (3,3)

Fall 4.2: $\exists$ gemeinsamer Nachbar d zweier Nachbarn von v, welcher verschieden von v ist. O.E. sei d Nachbar von a und b.
$\rightsquigarrow$ Nimm entweder $N(v) = \{a, b, c\}$ oder $\{d, v\}$ in das Vertex Cover.
Denn: Nimmt man v und nicht d ins VC, so müßte man a und b nehmen; dann wäre aber wiederum $N(v)$ schon mind. so gut gewesen.
Verzweigungsvektor: (3,2)

Fall 4.3: $\nexists$ Kante zwischen a, b, c und einer der Nachbarn in $N(v)$ habe Grad ≥ 4. Sei dies o.E. a mit $N(a) = \{v, a_1, a_2, a_3\}$.
$\rightsquigarrow$ Nimm entweder $N(v) = \{a, b, c\}$ oder $N(a) = \{v, a_1, a_2, a_3\}$ oder $\{a\} \cup N(b) \cup N(c)$ in das Vertex Cover.
Denn: Nimmt man v und a, so braucht man b oder c nicht mehr hinzuzunehmen, denn z.B. $\{v, a, b\}$ wäre auf keinen Fall besser als $\{a, b, c\}$.
Verzweigungsvektor: (3,4,6) (Beachte hierbei, daß $|\{a\} \cup N(b) \cup N(c)| \geq 6$, denn wegen Fall 4.2 gilt $N(b) \cap N(c) = \{v\}$ und damit

$|N(b) \cup N(c)| \geq 5$. Außerdem gilt $a \notin N(b)$ und $a \notin N(c)$ wegen Fall 4.1.)

Fall 4.4: sonst, d.h. es gibt keine Kante zwischen Elementen aus $N(v)$ und alle Elemente aus $N(v)$ besitzen genau Grad 3.
$\rightsquigarrow$ Nimm entweder $N(v) = \{a, b, c\}$ oder $N(a) = \{v, a_1, a_2\}$ oder $N(b) \cup N(c) \cup N(a_1) \cup N(a_2)$ in das Vertex Cover.
Denn: Im letzten „Zweig“ muß man a und v nehmen. Dann macht es „keinen Sinn“ mehr, b und c zu nehmen, also wähle $N(b) \cup N(c)$. Analog argumentiert man mit vertauschten Rollen für a und v, was zur Aufnahme von $N(a_1) \cup N(a_2)$ führt.
Verzweigungsvektor: (3,3,6) (Beachte hierbei, daß $N(b) \cap N(c) = \{v\}$, also $|N(b) \cup N(c)| = 5$, und es ist $a \in N(a_1)$ und $a \notin N(b) \cup N(c)$.)

Fall 5: Der Graph ist 4-regulär, d.h. jeder Knoten hat genau 4 Nachbarn.
$\rightsquigarrow$ Wähle beliebigen Knoten v, nimm entweder v oder $N(v)$ in das VC.
Verzweigungsvektor: (1,4).
Die zentrale Beobachtung hierbei ist es, daß dieser Fall nur *einmalig* während des Ablaufs des gesamten Suchbaum-Algorithmus auftritt. Wird nämlich nach v bzw. $N(v)$ verzweigt, so müssen die entstandenen Teilgraphen immer mind. einen Knoten vom Grad ≤ 3 enthalten. Dieser Fall spielt also asymptotisch keine Rolle.

Damit haben wir eine vollständige Fallunterscheidung. Zusammenfassend erhalten wir als Verbesserung von Satz 4.1.

Satz 4.12. *Vertex Cover kann in Zeit $O(kn + (1.342)^k \cdot k^2)$ gelöst werden.*

Beweis.
Bis auf die Größe des neuen Suchbaum schließt man analog zum Nachweis der Zeitkomplexität in Satz 4.1. Es bleibt also die Größe des neuen Suchbaums abzuschätzen. Diese ist durch die zu den jeweiligen Fällen gehörige Rekursionsgleichungen bestimmt. Diese wiederum sind eindeutig aus den Verzweigungsvektoren zu errechnen.

Wir finden

Fall	Verzweigungs-vektor	errechnete Basis der Baumgröße
1	-	-
2	(1,5)	1.325
3.1	-	-
3.2	-	-
3.3	(2,3)	1.325
4.1	(3,3)	1.260
4.2	(3,2)	1.325
4.3	(3,4,6)	1.305
4.4	(3,3,6)	1.342
5	(1,4)	1.381

Da wir wegen des maximal einmaligen Auftretens von Fall 5 diesen für die asymptotische Zeitkomplexität vernachlässigen können, liefert Fall 4.4 den worst case mit einer Baumgröße von $(1.342)^k$. □

Bemerkung 4.13.

- In Satz 4.12 (und auch allgemeiner für Methoden bestehend aus Reduktion auf den Problemkern und Suchbaum) läßt sich durch einfache Modifikation am Algorithmus und verbesserter mathematischer Analyse der Faktor k^2 durch eine kleine Konstante ersetzen (vgl.[21]). ertex Cover ist demgemäß sogar in Zeit $O(kn+(1.342)^k)$ lösbar. Analog kann man in Satz 4.11 den Faktor k^3 verschwinden lassen.

- Die besten derzeit bekannten parametrisierten Algorithmen für Vertex Cover erreichen eine Suchbaumgröße deutlich unter $(1.3)^k$ (vgl. hierzu [22, 27, 8]).

- Der beste „nicht-parametrisierte“, exakte Algorithmus für Vertex Cover hat Laufzeit $O(1.211^n)$, wobei n die Anzahl der Knoten im Graphen G ist (vgl. [25]).

4.3 Farbkodierungen und Hashing

In diesem Abschnitt besprechen wir eine Methode, die nicht mehr so „elementar“ ist wie z.B. die Suchbaummethode, jedoch mitunter fast vergleichbar gute Laufzeitwerte (exponentieller Faktor!) liefert.

Zur Illustration betrachten wir folgendes Problem, das im allgemeinen *NP*-vollständig ist (vgl. [15]):

Longest Path:
Gegeben: Ein ungerichteter Graph $G = (V, E)$ und ein $k \in \mathbb{N}$.
Frage: Gibt es einen *einfachen* Pfad der Länge $k - 1$ in G, d.h. einen Pfad

bestehend aus k Knoten, so daß kein Knoten zwei oder mehrmals auf dem Pfad auftaucht?

Bemerkung 4.14.
- Eine Variante des obigen Problems, das sogenannte *Longest Cycle*-Problem, welches ebenfalls zu den NP-vollständigen Problemen gehört, wäre im folgenden analog zu behandeln.
- Über das Bilden der k-ten Potenz der sogenannten Adjazenzmatrix lassen sich (mit algebraischen Mitteln) all die Knotenpaare zu finden, welche durch Pfade der Länge k miteinander verbunden sind. In der Regel sind die so gefundenen Pfade allerdings nicht einfach.

Wir beschreiben zunächst einen randomisierten Ansatz zur Lösung von Longest Path:

Färbe die Knoten von G per Zufall mit k verschiedenen Farben. Wir nennen einen Pfad *voll bunt*, wenn jeder seiner Knoten mit einer anderen Farbe gefärbt wurde.

Offensichtlich ist jeder voll bunte Pfad einfach. Umgekehrt gilt: Jeder einfache Pfad der Länge $k-1$ ist mit Wahrscheinlichkeit $\frac{k!}{k^k} > e^{-k}$ voll bunt.

Lemma 4.15. *Es sei $G = (V, E)$ ein Graph und $c : V \to \{1, \ldots, k\}$ sei eine Färbung seiner Knoten mit k verschiedenen Farben. Dann kann ein voll bunter Pfad der Länge $k-1$ in G (falls ein solcher existiert) in Zeit $2^{O(k)} \cdot |E|$ gefunden werden.*

Beweis.
Nachfolgend beschreiben wir einen Algorithmus, der alle voll bunten Pfade der Länge $k-1$ ausgehend von einem Startknoten s findet. Dies ist keine Einschränkung, denn um das allgemeine Problem zu lösen, fügen wir einfach einen neuen Knoten s' zu V hinzu, färben ihn mit der neuen Farbe 0 und verbinden ihn mittels Kanten zu jedem Knoten aus V.

Der nachfolgende Ansatz beruht auf der Methode des *Dynamischen Programmierens.*
Annahme: $\forall\, v \in V$ wurden bereits alle möglichen Farbmengen von voll bunten Pfaden zwischen s und v mit Länge i gefunden.
Wichtig: Nicht die Pfade, sondern nur die Farbmengen werden gespeichert. Für jeden Knoten v gibt es jeweils höchstens $\binom{k}{i}$ viele solcher Mengen.
Sei nun C eine solche Farbmenge, die zu v gehört. Wir betrachten jedes zu v gehörende C und jede Kante $\{u, v\} \in E$: Füge $c(u)$ zu C hinzu, falls $c(u) \notin C$. So erhalten wir alle möglichen Farbmengen, die zu Pfaden der Länge $i+1$ gehören, usw. Damit erhält G einen voll bunten Pfad bezüglich der Färbung c, genau dann wenn es einen Knoten $v \in V$ gibt, der mindestens eine Farbmenge besitzt, die zu einem Pfad der Länge $k-1$ korrespondiert.

Zeitkomplexität: Der beschriebene Algorithmus führt $O(\sum_{i=1}^{k} i\binom{k}{i} \cdot |E|)$ Operationen aus. Dabei entspricht der Faktor i jeweils dem Test, ob $c(u)$ schon in C liegt. Der Faktor $\binom{k}{i}$ gibt die Anzahl der möglichen C-Mengen an, und der Faktor $|E|$ wird benötigt, um den Test $\{u, v\} \in E$ durchzuführen. Der gesamte Ausdruck läßt sich, wie behauptet, durch $O(k\, 2^k |E|)$ abschätzen. □

Aufgabe 4.16. Wie wird im obigen Beweis der gesuchte Pfad tatsächlich konstruiert?

Satz 4.17. *Longest Path kann in „erwarteter" Laufzeit $2^{O(k)} \cdot |E|$ gelöst werden.*

Beweis.
Nach obigen Vorbemerkungen ist mit Wahrscheinlichkeit $\frac{k!}{k^k} > e^{-k}$ ein einfacher Pfad der Länge $k-1$ voll bunt. Nach Lemma 4.15 kann ein solcher voll bunter Pfad in Zeit $2^{O(k)} \cdot |E|$ gefunden werden; genauer ergibt sich hieraus auch, daß alle voll bunten Pfade der Länge $k-1$ gefunden werden können.

Wiederholen wir also folgenden Vorgang $e^k = 2^{O(k)}$ mal:

1. Wähle per Zufall eine Knotenfärbung $c : V \to \{1, \dots, k\}$.
2. Überprüfe mit Lemma 4.15, ob es einen voll bunten Pfad gibt; falls ja, so ist dies ein einfacher Pfad der Länge $k-1$.

Nach e^k Wiederholungen ist der Erwartungswert für die gefundenen voll bunten Pfade (falls solche existieren) größer als $e^k \cdot e^{-k} = 1$. Also kann Longest Path in erwarteter Laufzeit $e^k \cdot 2^{O(k)} \cdot |E| = 2^{O(k)} \cdot |E|$ gelöst werden. □

Der in Satz 4.17 beschriebene Algorithmus ist randomisiert („erwartete Laufzeit"). Mit Hilfe von Hashing kann er mit geringen Effizienzeinbußen in einen deterministischen umgewandelt werden.

Zur Erinnerung rekapitulieren wir die Grundidee beim Hashing. Ziel dieser Methode ist es, „wenige" Elemente eines „großen Raums" eindeutig auf einen „kleinen Raum" abzubilden. Anwendungen dieser Technik finden sich z.B. im Compilerbau, wo etwa das Anlegen einer Symboltabelle für die Variablen eines Programms über Hashtabellen geregelt wird.

Für unsere Zwecke benötigen wir folgendes:
Eine Liste von Färbungen der Knoten V eines Graphen, so daß für *jede* Teilmenge $V' \subseteq V$ der Größe k gilt: Es gibt mindestens eine Färbung in dieser Liste derart, daß jeder Knoten in V' eine andere Farbe hat.

Dies läßt sich mit dem Begriff der k-perfekten Familien von Hash-Funktionen von $\{1, 2, \ldots, |V|\}$ nach $\{1, 2, \ldots, k\}$ formalisieren:

Definition 4.18. Eine *k-perfekte Familie von Hash-Funktionen* ist eine Familie $\mathcal{F}$ von Funktionen von $\{1, \ldots, n\}$ nach $\{1, \ldots, k\}$ derart, daß zu jeder Teilmenge $S \subseteq \{1, \ldots, n\}$ mit $|S| = k$ ein $f \in \mathcal{F}$ existiert, so daß f auf S bijektiv ist.

Mittels tiefliegender mathematischer Methoden, welche den Rahmen dieser Vorlesung sprengen würden, läßt sich zeigen (vgl. etwa [1]):

Mitteilung 4.19. *Es ist möglich, Familien k-perfekter Hash-Funktionen von $\{1, \ldots, n\}$ nach $\{1, \ldots, k\}$ zu konstruieren, die aus $2^{O(k)} \log n$ vielen Elementen bestehen.*

Hinweis 4.20. Für ein f aus der in Mitteilung 4.19 beschriebenen Familie k-perfekter Hash-Funktionen läßt sich $f(i)$ (mit $1 \leq i \leq n$) in linearer Zeit berechnen.

Satz 4.21. *Longest Path kann deterministisch in Laufzeit $2^{O(k)} \cdot |E| \cdot \log |V|$ gelöst werden. Insbesondere ist also Longest Path in FPT.*

Beweis.
Färbe den Graph mittels aller möglicher Hash-Funktionen aus der in Mitteilung 4.19 gegebenen Familie. Nach Definition 4.18 muß bei mindestens einer dieser Färbungen ein einfacher Pfad voll bunt werden. Ein voll bunter Pfad läßt sich dann wieder mit Lemma 4.15 finden.

Da die Familie aus Mitteilung 4.19 aus $2^{O(k)} \log n$ vielen Elementen besteht, muß die Komplexität des Algorithmus in Lemma 4.15 mit diesem Faktor multipliziert werden. Somit erhalten wir die Gesamtlaufzeit

$$2^{O(k)} \log n \cdot 2^{O(k)}|E| = 2^{O(k)}|E| \log |V|,$$

wo $n = |V|$. □

Hashing allein schon kann helfen, *FPT*-Algorithmen anzugeben:

Zur Illustration betrachten wir das im allgemeinen *NP*-vollständige *Multidimensional Matching* Problem:

Gegeben: Eine Menge $M \subseteq X_1 \times \ldots \times X_r$, wobei X_i paarweise disjunkte Mengen sind und $k \in \mathbb{N}$.

Frage: Existiert $M' \subseteq M$ mit $|M'| = k$, so daß keine zwei Elemente in irgendeiner Koordinate übereinstimmen?

Satz 4.22. *Multidimensional Matching kann in Zeit* $O(2^{O(kr)}(kr)!\,nr\log^2 n)$ *gelöst werden, d.h. Multidimensional Matching ist in FPT.*

Beweis.
Sei N die Anzahl aller verschiedenen Koordinaten, die in der Eingabe M auftreten. Ohne Einschränkung läßt sich also die Menge aller Koordinatenwerte durch $\{1, \dots, N\}$ repräsentieren. Desweiteren setze $K := kr$. Schließlich bezeichne n die Gesamtgröße der Beschreibung von M, sprich die Eingabegröße.

Definiere ein *Lösungsschema* S als eine Menge von k vielen r-Tupeln, so daß die Vereinigung aller Koordinaten genau $\{1, \dots, K\}$ ergibt.

Sein nun $\mathcal{H}(N, K)$ eine Familie k-perfekter Hash-Funktionen von $\{1, \dots, N\}$ nach $\{1, \dots, K\}$ (vgl. Definition 4.18 und Mitteilung 4.19).

Folgender Algorithmus löst das Multidimensional Matching Problem:

> **for** alle $h \in \mathcal{H}(N, K)$ und alle Lösungsschemata S **do**
>   **for** alle $\alpha \in M$ mit $\alpha = (\alpha_1, \dots, \alpha_r),\ \alpha_i \in \{1, \dots N\}$ **do**
>   berechne $(h(\alpha_1), \dots, h(\alpha_r))$;
>   **if** jedes r-Tupel in S erscheint als Bild $(h(\alpha_1), \dots, h(\alpha_r))$
>   für ein $\alpha = (\alpha_1, \dots, \alpha_r)$
>   **then** Multidimensional Matching hat eine Lösung
>   bestehend aus „diesen“ α's;
> **if** in voriger Schleife wurde keine Lösung gefunden
>   **then** Multidimensional Matching hat keine Lösung;

Zur Korrektheit des Algorithmus: Falls M ein „k-Matching“ besitzt, dann beinhaltet das $K = kr$ verschiedene Koordinaten und nach Mitteilung 4.19 existiert ein $h \in \mathcal{H}(N, K)$, so daß diese bijektiv auf $\{1, \dots K\}$ abgebildet werden; das Bild unter h ist dann ein gesuchtes Lösungsschema S.
Existiert umgekehrt ein h, welches ein Lösungsschema wie beschrieben als Bild liefert, dann ergibt das Urbild unter h wiederum ein „k-Matching“.

Zur Laufzeit des Algorithmus:

- Nach Mitteilung 4.19 besteht $\mathcal{H}(N, K)$ aus $2^{O(K)} \cdot \log N$ vielen Hashfunktionen.
- Es gibt $K!$ Möglichkeiten für Lösungsschemata.
- Es gibt $O(n)$ viele Elemente α.
- Die Berechnung von $(h(\alpha_1), \dots, h(\alpha_r))$ läßt sich in Zeit $O(r \log N)$ durchführen.

Zusammen erhalten wir also die behauptete Laufzeit

$$O(2^{O(K)} \cdot \log N \cdot K! \cdot n \cdot r \log N) = O(2^{O(K)} \cdot K! \cdot n \cdot r \cdot \log^2 n).$$

$\square$

Wir wollen abschließend zur Hashing-Methode bemerken, daß die exponentielle Komponente des parametrisierten Algorithmus in der Regel (auch) von der Größe der Hashfamilie abhängt.

4.4 Aufwendigere Methoden im Überblick

In diesem Abschnitt sollen kurz aufwendigere Methoden zur Gewinnung parametrisierter Algorithmen dargestellt werden. Eine halbwegs erschöpfende Darstellung dieser Methoden bedürfte wohl jeweils für sich einer eigenen Vorlesung. Darüberhinaus sind die hier vorgestellten Methoden eher von theoretischem Interesse, da sie bislang offenbar noch nicht zu *effizienten* parametrisierten Algorithmen führen. Nachfolgend beschränken wir uns auf die zwei bekanntesten Ansätze, beruhend auf Graphminoren und die Baumweite von Graphen.

4.4.1 Graphminoren

Hierbei handelt es sich im wesentlich um die Verallgemeinerung eines Satzes von Kuratowski, welcher besagt, daß ein Graph genau dann planar ist, wenn er nicht die Minoren $K_{3,3}$ (vollständiger bipartiter Graph der Größe 3,3) und K_5 (vollständiger Graph der Größe 5) besitzt.

Definition 4.23. Ein Graph H heißt **Minor** eines Graphen G (in Zeichen $H \leq_m G$), falls ein zu H „isomorpher“ Graph durch wiederholte Anwendung folgender zwei Operationen aus G erzeugt werden kann:

i) Herausnehmen eines Teilgraphen.

ii) Kantenkontraktion, d.h. ersetze eine Kante $\{u, v\}$ durch einen neuen Knoten w derart, daß w Kanten zu allen Nachbarn von u und v erhält (wobei u und v selbst gelöscht werden).

Eine Familie $\mathcal{F}$ von Graphen heißt **abgeschlossen unter Minorenbildung**, falls gilt: Wenn $G \in \mathcal{F}$ und $H \leq_m G$, dann gilt $H \in \mathcal{F}$.

Aufgabe 4.24. Man versuche, den Teilgraphen H in Abbildung 6 als Minor des Graphen G zu identifizieren.

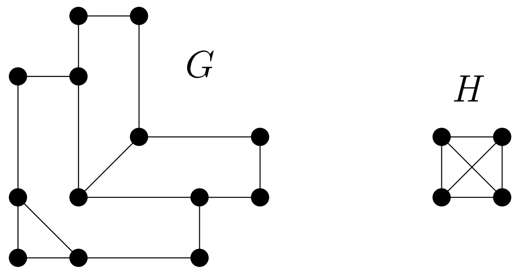


Abbildung 6: Minorenbildung aus gegebenem Graphen.

Zentral für die Bedeutung von Graphenminoren sind folgende zwei Sätze.

Mitteilung 4.25. *Es sei $\mathcal{F}$ eine Familie von endlichen Graphen, die abgeschlossen unter Minorenbildung ist, dann existiert eine endliche „Obstruktionsmenge“ von Graphen $O_{\mathcal{F}} = \{H_1, \dots H_t\}$, so daß*

$$G \notin \mathcal{F} \quad \Leftrightarrow \quad \exists\, i \in \{1, \dots, t\} : \;\; H_i \leq_m G$$

gilt.

Der eingangs zitierte Satz von Kuratowski ist also ein Spezialfall dieses Satzes, bei welchem die Obstruktionsmenge explizit angegeben werden kann. Beachte, daß dies im allgemeinen nicht der Fall zu sein braucht. Es handelt sich um eine reine Existenzaussage. Insbesondere existiert kein allgemeiner Algorithmus zur Konstruktion einer solchen Obstruktionsmenge.

Der Satz aus Mitteilung 4.25 wird auch „Garphminorensatz“ genannt und geht auf N. Robertson und P. D. Seymour zurück. Der überaus tiefliegende Beweis des Satzes ist über eine Serie von mehr als 20 Artikeln verteilt, welche seit den 80er Jahren veröffentlicht wurden.

Die wichtigsten algorithmischen Konsequenzen des Graphminorensatzes ergeben sich aus folgendem — ebenfalls Robertson/Seymour zu verdankenden — „Minorentest“:

Mitteilung 4.26. *Nachfolgender Minorentest kann für einen festen Graphen H („der Parameter“) in Laufzeit $O(n^3)$ (genauer: $O(f(k) \cdot n^3)$, wo $k \leq$ Größe von H) gelöst werden, das Problem ist also in FPT:*

Gegeben: *Graphen $G = (V, E)$ und $H = (V', E')$.*
Frage: *Gilt $H \leq_m G$, d.h. ist H Minor von G?*

Die Konstante, die sich hinter der O-Notation in Mitteilung 4.26 verbirgt, hängt von dem Parameter H ab und ist riesig. Mit Mitteilung 4.26 lassen sich also leider keine effizienten parametrisierten Algorithmen gewinnen. Das

Ergebnis kann allerdings als ein „Klassifikationswerkzeug“ verwendet werden.

Als typisches Anwendungsbeispiel für die Verwendung des Graphminorensatzes betrachten wir erneut Vertex Cover:
Es ist leicht nachzuprüfen, daß für festes k die Familie von Graphen, welche ein Vertex Cover der Größe $\leq k$ besitzen, abgeschlossen ist unter Minorenbildung. Nach dem Graphminorensatz (Mitteilung 4.25) gibt es also eine endliche Obstruktionsfamilie O_k. Für einen gegebenen Graphen G kann mithilfe des Minorentests (Mitteilung 4.26) somit überprüft werden, ob für mindestens ein Element $H \in O_k$ gilt, daß $H \leq_m G$, sprich ob G ein Vertex Cover der Größe $\leq k$ besitzt. Auf diese Weise erhalten wir einen $O(f(k) \cdot n^3)$-Algorithmus für Vertex Cover. Dabei ist f eine Funktion, welche durch die Größe der Graphen der endlichen Obstruktionsmenge O_k bestimmt wird. Insgesamt zeigt dies, daß Vertex Cover in FPT liegt.

Die erste Veröffentlichung, in welcher nachgewiesen wurde, daß Vertex Cover in FPT liegt, beruhte in der Tat auf diesem Beweisprinzip. Man vergleiche diesen Algorithmus mit dem einfacheren und dennoch wesentlich effizienteren $O(2^k \cdot n)$-Suchbaumalgorithmus. Dies zeigt erneut, daß die Verwendung des Graphenminorensatzes in der Regel nicht zur Entwicklung effizienter Algorithmen, sondern eher als „Klassifikationswerkzeug“ verstanden werden muß. Außerdem ist der Graphminorensatz (Mitteilung 4.25) inhärent nichtkonstruktiv, d.h. er liefert kein algorithmisches Verfahren zur Erzeugung der Obstruktionsmengen.

4.4.2 Beschränkte Baumweiten

Grundlegend für den folgenden Ansatz ist die Definition der Baumweite (vgl. Definition 2.4).

Die entscheidende Beobachtung ist nun:
Viele NP-vollständige Probleme können in Polynomzeit oder sogar in linearer Laufzeit gelöst werden, wenn sie auf Graphen mit konstanter Baumweite beschränkt werden; insbesondere ist dies der Fall, wenn die zugehörige Baumzerlegung bekannt ist. Wird die Baumweite durch einen Parameter k beschränkt, so erhält man Algorithmen mit Laufzeiten der Form $f(k) \cdot n^{O(1)}$, sprich FPT-Algorithmen.

Mitteilung 4.27. *Es gibt einen Algorithmus mit Laufzeit $O(f(k) \cdot n)$, welcher für einen gegebenen Graphen überprüft, ob er Baumweite höchstens k besitzt. Falls ja, so liefert der Algorithmus darüber hinaus auch eine zugehörige Baumzerlegung.*

Es sollte angemerkt werden, daß der zu Mitteilung 4.27 gehörige Algorithmus (siehe [5]) selbst sehr aufwendig ist und große konstante Faktoren mit sich bringt. Es ist ein aktuelles Forschungsthema, die Effizienz eines solchen Algorithmus zu verbessern.

Mit diesem Resultat ergibt sich folgender Grundansatz zur Gewinnung von FPT-Algorithmen:

1. Mithilfe des Algorithums aus Mitteilung 4.27 ermittle die Baumzerlegung des Eingabegraphen.

2. Wende einen Polynomzeitalgorithmus an, welcher auf der Baumzerlegung beruht.

Wichtige Bemerkung hierzu ist, daß für viele parametrisierte Probleme gilt, daß die „Ja-Instanzen" (oder manchmal die „Nein-Instanzen") eine durch den Parameter beschränkte Baumweite haben. Damit kann nach obigem Schritt 1 gleich entschieden werden, ob Schritt 2 noch auszuführen ist.
Schritt 2 erfordert in der Regel problemspezifische, komplizierte Algorithmen; eine Vorstellung eines solchen würde den Rahmen der Vorlesung sprengen.

Zum Schluß des Abschnitts wollen wir nochmals eine „Intuition für die Grundidee der Baumzerlegung" geben: Es ist hilfreich sich vorzustellen, daß eine Baumzerlegung eine Art „Landkarte" der Struktur (und Analyse) eines Graphens darstellt. Oftmals geben die Baumknoten der Zerlegung — diese entsprechen einer Menge von Knoten des ursprünglichen Eingabegraphen — an, welche Information ein Algorithmus jeweils „halten" muß. Als Schlagwörter seien hier etwa „Parse-Bäume" oder „Baum-Automaten" gennannt.

Beispielsweise besagt ein Satz von Courcelle [10], daß jede Grapheneigenschaft, die sich in sog. *monadischer Logik 2.Stufe* ausdrücken läßt, für beschränkte Baumweiten erkennbar ist. Genauer bedeutet „erkennbar", daß im Fall von Eingabegraphen mit beschränkten Baumweiten ein Baumautomat angegeben werden kann, welcher erkennt, ob der Graph die besagte Eigenschaft hat. Auf diese Weise kann nach obigem Schema ein FPT-Algorithmus für derartige Eigenschaften konstruiert werden.

5 Bezüge zur Approximation

Es gibt enge Bezüge zwischen der Approximierbarkeit von Problemen und der Existenz von FPT-Algorithmen. Zunächst geben wir einige grundlegende Definitionen aus dem Bereich der Approximationsalgorithmen.

Definition 5.1. Ein *NP-Optimierungsproblem* Q ist entweder ein Maximierungs- oder ein Minimierungsproblem, welches gegeben ist durch ein 4-Tupel (I_Q, S_Q, f_Q, opt_Q) mit

(i) I_Q ist die Menge der *Eingabeinstanzen*, erkennbar in polynomieller Zeit.

(ii) $S_Q(x)$ ist die Menge der *zulässigen Lösungen* auf Eingabe $x \in I_Q$, so daß es ein Polynom p und eine in Polynomzeit berechenbare Relation Π gibt derart, daß

$$\forall\, x \in I_Q : \; S_Q(x) = \{ y : |y| \leq p(|x|) \text{ und } \Pi(x,y) \}$$

(iii) $f_Q(x,y) \in \mathbb{N}$ ist die *Zielfunktion* für jedes $x \in I_Q$ und $y \in S_Q(x)$; und $f_Q(x,y)$ ist in Polynomzeit berechenbar.

(iv) $\mathrm{opt}_Q \in \{\max, \min\}$.

Eine *optimale Lösung* für Eingabe $x \in I_Q$ ist definiert als

$$opt_Q(x) := opt_Q\{ f_Q(x,z) : z \in S_Q(x) \}.$$

Beispiel 5.2. Man mache sich Definition 5.1 am Beispiel $Q =$Vertex Cover klar: I_Q ist hier die Menge aller ungerichteten Graphen, x entspricht einem solchen Graphen, $S_Q(x)$ besteht aus allen Knotenmengen, die ein (nicht notwendig minimales) Vertex Cover bilden, $f_Q(x,y)$ entspricht der Anzahl der Knoten in der Vertex Cover Menge y und $opt_Q(x) = \min$.

Zu einem $NP-$Optimierungsproblem Q läßt sich in natürlicher Weise eine „*parametrisierte Version*" angeben:

Gegeben: $x \in I_Q$ und $k \in \mathbb{N}$.
Frage a): $\exists\, y \in S_Q(x)$ mit $f_Q(x,y) \geq k$? (Maximierungsproblem)
Frage b): $\exists\, y \in S_Q(x)$ mit $f_Q(x,y) \leq k$? (Minimierungsproblem)

Definition 5.3. 1. Ein NP-Optimierungsproblem (I_Q, S_Q, f_Q, opt_Q) hat einen *ε-Approximationsalgorithmus*, falls es für jedes $x \in I_Q$ einen Polynomzeit-Algorithmus gibt, der ein $y \in S_Q(x)$ ausgibt mit der Eigenschaft

$$\frac{|f_Q(x,y) - \mathrm{opt}_Q(x)|}{\max\{\mathrm{opt}_Q(x), f_Q(x,y)\}} \;\leq\; \varepsilon.$$

2. Ein *NP*-Optimierungsproblem Q hat ein *Polynomzeit-Approximationsschema* (polynomial time approximation scheme, kurz: PTAS), wenn es für jedes $\varepsilon > 0$ und jedes $x \in I_Q$ einen Polynomzeit ε-Approximationsalgorithmus gibt.

3. Ein PTAS heißt *volles Polynomzeit-Approximationsschema* (fully polynomial time approximation scheme, kurz: FPTAS), falls die Laufzeit des ε-Approximationsalgorithmus polynomiell sowohl bezüglich der Eingabegröße $|x|$ als auch bezüglich $\frac{1}{\varepsilon}$ ist.

Beispiel 5.4. Für Einzelheiten siehe z.B. das Buch von Papadimitriou [23].

1. Gemäß Kapitel 1 (vgl. die Faktor-2-Approximation aus Lemma 1.3) besitzt das Vertex Cover Problem einen $\frac{1}{2}$-Approximationsalgorithmus.

2. **Bin Packing:**
   Gegeben: $a_1, \ldots, a_N, C, B \in \mathbb{N} \setminus \{0\}$.
   Frage: Können die Zahlen $a_1, \ldots, a_N$ in B Teilmengen so unterteilt werden, daß für jede die Summe ihrer Elemente höchstens C ist?

   Dieses Problem ist *NP*-vollständig. Es besitzt ein PTAS, nicht jedoch eine FPTAS (es sei denn $P = NP$). Die Laufzeit ist $n^{O(1/\epsilon)}$.

3. **Knapsack:**
   Gegeben: $s_1, \ldots, s_N$, $v_1, \ldots, v_N$, $W, K \in \mathbb{N} \setminus \{0\}$.
   Frage: $\exists\, T \subseteq \{1, \ldots, N\}$, so daß $\sum_{i \in T} s_i \leq W$ und $\sum_{i \in T} v_i \geq K$?

   Dieses Problem ist ebenso *NP*-vollständig. Knapsack besitzt jedoch ein FPTAS — der Algorithmus hat Laufzeit $O(\frac{n^3}{\varepsilon})$.

Folgendes, einfaches Ergebnis stellt eine wichtige Beziehung zwischen vollen Polynomzeitschemata (FPTAS) und der parametrisierten Klasse *FPT* her:

Satz 5.5. *Falls ein NP-Optimierungsproblem ein FPTAS besitzt, dann ist die parametrisierte Version des Problems in FPT.*

Beweis.
Sei $Q = (I_Q, S_Q, f_Q, opt_Q)$ ein *NP*-Optimierungsproblem mit einem FPTAS. Wir beschränken uns in den nachfolgenden Betrachtungen auf Maximierungsprobleme, d.h. auf den Fall $opt_Q = \max$; der Beweis für den Minimierungsfall wird analog geführt.

Sei weiter Q_k die parametrisierte Version von Q: Gegeben $x \in I_Q$ und $k \in \mathbb{N}$; existiert $y \in S_Q(x)$ mit $f_Q(x, y) \geq k$?

Wir wählen $\varepsilon = \frac{1}{2k}$. Da Q ein FPTAS besitzt, gibt es also einen Algorithmus mit polynomieller Laufzeit in der Eingabegröße $n = |x|$ und $\frac{1}{\varepsilon} = 2k$, der für Eingabe $x \in I_Q$ ein $y \in S_Q(x)$ ausgibt mit

$$\frac{\max_Q(x) - f_Q(x,y)}{\max_Q(x)} \leq \varepsilon = \frac{1}{2k}.$$

Arithmetische Umformung ergibt

$$\frac{f_Q(x,y)}{\max_Q(x)} \geq 1 - \frac{1}{2k} = \frac{2k-1}{2k}.$$

Im Falle $\max_Q(x) \geq k$ folgt hiermit $f_Q(x,y) \geq k - \frac{1}{2}$. Da $f_Q(x,y)$ ganzzahlig ist, folgt somit auch $f_Q(x,y) \geq k$.

Wegen der Beziehung $f_Q(x,y) \leq \max_Q(x)$ folgt im Fall $\max_Q(x) < k$ erst recht $f_Q(x,y) < k$.

Somit ergibt sich mit der Wahl von $\varepsilon = \frac{1}{2k}$ ein parametrisierter Algorithmus, der zeigt, daß das parametrisierte Problem in FPT liegt. □

Wir schließen diesem Satz zwei wichtige Bemerkungen an.

Bemerkung 5.6.
1. Der im Beweis zu Satz 5.5 beschriebene Algorithmus hat eine Laufzeit, die polynomiell sowohl in $n = |x|$ als auch $2k$ ist, also durch ein Polynom $p(n,k)$ in zwei Variablen beschrieben werden kann. Die Definition von FPT (Definition 3.7) erlaubt aber sogar Laufzeiten der Form $f(k) \cdot n^{O(1)}$ — sie müssen also nicht polynomiell in k sein. Demzufolge könnte man in Satz 5.5 die Forderung nach einem FPTAS ersetzen durch die Forderung nach einem PTAS mit der zusätzlichen Eigenschaft, daß es eine Konstante c unabhängig von ε gibt, so daß der ε-Approximationsalgorithmus für *festes* $\varepsilon > 0$ in Zeit $O(n^c)$ läuft.

2. Die Umkehrung von Satz 5.5 (d.h. die Implikation „Nicht $FPT \Rightarrow$ kein FPTAS“) liefert die interessantere Aussage: Läßt sich für ein parametrisiertes Problem „zeigen“, daß wenn es („wahrscheinlich“) nicht in FPT liegt, so existiert (wahrscheinlich) auch kein FPTAS für das zugehörige Optimierungsproblem. Mehr dazu im nächsten Kapitel.

Abschließend sei noch bemerkt, daß sich für bestimmte, „syntaktisch zu definierende“ NP-Optimierungsprobleme zeigen läßt, daß ihre parametrisierten Versionen in FPT liegen. Genauer heißt das, daß alle Probleme aus der sogenannten Maximierungsklasse MAXSNP und der Minimierungsklasse $\mathrm{MINF}^+\Pi_1(h), h \geq 2$, in FPT sind [7]. Ein Beispielproblem aus MAXSNP ist MaxSat, eines aus $\mathrm{MINF}^+\Pi_1(h)$ ist Vertex Cover. Das Problem MaxSat ist sogar MAXSNP-vollständig. Für derartige Probleme ist z.B. bekannt, daß sie sich bis auf einen bestimmten konstanten Faktor approximieren lassen (es existiert also ein ε-Approximationsalgorithmus mit $\varepsilon < 1$), aber sie besitzen kein FPTAS, wenn nicht $P = NP$ (vgl. [2]).

Es sei angemerkt, daß mittels solchen allgemeinen Aussagen (wie Satz 5.5) in der Regel nicht die effizientesten parametrisierten Algorithmen erhalten werden können.

6 Parametrisierte Komplexitätstheorie

Nicht für alle parametrisierten Probleme lassen sich FPT-Algorithmen angeben. Ein Beweis dieser Aussage ist jedoch jenseits der gegenwärtigen Möglichkeiten der Komplexitätstheorie. Allerdings hilft auch hier (wie bei der „klassischen Komplexitätstheorie“ mit der P versus NP Problematik) der Übergang von „absoluter“ zu „relativer“ Komplexität. Mithilfe von Begriffen wie Reduktion und Vollständigkeit lassen sich „vollständige Probleme“ definieren, so daß viele Indizien dagegen sprechen, daß diese Probleme in FPT liegen. Beispiele solcher Probleme sind u.a. Independent Set und Dominating Set.

Im folgenden wollen wir die notwendigen Begriffe und Formalismen entwickeln, die zu einer solchen Theorie „hartnäckiger“ parametrisierter Probleme führen. Der Begriff der Reduktion ist dabei das Herzstück.

Es sei angemerkt, daß wir hier natürlich nur eine stark verkürzte und vereinfachte Darstellung der Ergebnisse geben können.

6.1 Parametrisierte Reduktion

Um parametrisierte Probleme sinnvoll klassifizieren zu können, benötigen wir den Begriff der Reduktion. Dieser erweist sich als technisch schwieriger (da „feiner“) als der in der klassischen Komplexitätstheorie übliche Reduktionsbegriff. Nachfolgend geben wir die wohl wichtigste Definition im Rahmen der parametrisierten Komplexitätstheorie.

Definition 6.1. Seien L und L' zwei parametrisierte Probleme über Σ. Wir sagen, daß es eine *parametrisierte Reduktion* von L auf L' gibt, wenn folgendes erfüllt ist: Es gibt Funktionen $k \mapsto k'$ und $k \mapsto k''$ auf $\mathbb{N}$ und eine Funktion $(x, k) \mapsto x'$ von $\Sigma^* \times \mathbb{N}$ nach Σ^* derart, daß

1. $(x, k) \mapsto x'$ ist in Zeit $k''|x|^{O(1)}$ berechenbar.
2. $(x, k) \in L$ gdw. $(x', k') \in L'$.

Beachte, daß in obiger Definition gefordert wird, daß weder k' noch k'' vom Eingabewort x abhängen. Lediglich x' darf von k und x abhängen. Außerdem sei darauf hingewiesen, daß es auch noch die feineren Konzepte der streng uniformen, uniformen und nicht-uniformen parametrisierten Reduktion gibt. Diese sind für uns allerdings hier ohne Belang.

Der Begriff der parametrisierten Reduktion soll an folgendem Beispiel ausführlich erläutert werden.

Beispiel 6.2. 1) Wir beschäftigen uns zunächst mit folgenden beiden Problemen:

Weighted CNF Sat:

Gegeben: Aussagenlogische Formel F in konjunktiver Normalform und ein Parameter $k \in \mathbb{N}$.
Frage: Existiert eine erfüllende Belegung vom *Gewicht* k, d.h. eine Belegung, bei der genau k Variablen mit „wahr" belegt werden?

Analog hierzu definiert man **Weighted 3CNF Sat**. Dabei darf jede Klausel aus höchstens 3 Literalen bestehen.

Weighted Binary Integer Programming:

Gegeben: Eine Matrix A und ein Vektor $\overrightarrow{b}$ mit Einträgen aus $\{0, 1\}$ und ein Parameter $k \in \mathbb{N}$.
Frage: Hat $A \cdot \overrightarrow{x} \geq \overrightarrow{b}$ einen Lösungsvektor über $\{0, 1\}$, der aus genau k Einsen besteht (dabei ist „$\geq$" komponentenweise zu verstehen)?

a) Um in der klassischen Komplexitätstheorie die *NP*-Vollständigkeit von 3CNF Sat (kurz: 3Sat) zu zeigen, reduziert man Sat auf 3Sat. Die Kernidee ist dabei wie folgt: Ersetze jeweils eine Klausel der Form $(l_1 \vee \ldots \vee l_m)$ durch einen Ausdruck der Form

$$(l_1 \vee l_2 \vee z_1) \wedge (\overline{z}_1 \vee l_3 \vee z_2) \wedge \ldots \wedge (\overline{z}_{m-3} \vee l_{m-1} \vee l_m). \qquad (2)$$

Hierbei bezeichnen $z_1, \ldots, z_{m-3}$ jeweils zusätzliche, neu eingeführte Variablen, die in der bisherigen Formel nicht vorkamen, und $l_1, \ldots, l_m$ sind Variablen, oder negierte Variablen, sprich Literale.

Es ist leicht nachzuprüfen, daß die solchermaßen entstehende 3CNF-Formel genau dann erfüllbar ist, wenn es die ursprüngliche CNF-Formel war; die Formeln sind also — wenn auch nicht „logisch äquivalent" — „erfüllbarkeitsäquivalent".

Diese Reduktion von Sat auf 3Sat ergibt aber keine parametrisierte Reduktion: Nehmen wir an, die ursprüngliche Sat-Formel hätte eine erfüllende Belegung mit Gewicht k, welche *genau ein* Literal l_j in der Klausel $(l_1 \vee \ldots \vee l_m)$ wahr macht. Um aber nun den Ausdruck (2) zu erfüllen, muß man alle Variablen $z_1, \ldots, z_{j-2}$ mit „wahr" belegen. Das bedeutet aber, daß sich das Gewicht der erfüllenden Belegung ändert, genauer gesagt, um $j - 2$ für diese Klausel größer wird. Entgegen Definition 6.1 hinge damit der Wert des „neuen" Parameters nicht nur vom „alten" ab, sondern auch von der Struktur der alten Formel (Klauselgrößen, ...). Also handelt es sich um keine parametrisierte Reduktion.

In der Tat hätte es schwerwiegende Konsequenzen (Zusammenbruch der noch einzuführenden „W-Hierarchie" von parametrisierten Komplexitätsklassen), gäbe es eine parametrisierte Reduktion von Weighted CNF Sat auf Weighted 3CNF Sat. Deshalb gilt die Existenz einer solchen parametrisierten Reduktion als unwahrscheinlich, ja der Nachweis einer solchen käme einer Sensation gleich.

b) Eine aussagenlogische Formel wird als *monoton* bezeichnet, falls sie keine Negationssymbole beinhaltet. Damit erhält man in natürlicher Weise das **Weigthed Monotone CNF Sat** Problem. Wir beschreiben nun eine parametrisierte Reduktion von Weighted Monotone CNF Sat auf Weighted Binary Integer Programming:

Gegeben sei die CNF-Formel $F = C_1 \wedge \ldots \wedge C_p$, wobei $C_1, \ldots, C_p$ Klauseln sind, die aus den Variablen $x_1, \ldots, x_m$ bestehen. Definiere die Binärmatrix $A = (a_{ij})_{1 \leq i \leq p, 1 \leq j \leq m}$ vermöge

$$a_{ij} := \begin{cases} 1 & \text{falls} \quad x_j \in C_i \\ 0 & \text{falls} \quad x_j \notin C_i \end{cases},$$

und sei $\overrightarrow{b}$ der aus p Einsen bestehende Vektor. Dann gilt:

$$A \cdot \overrightarrow{x} \geq \overrightarrow{b} \text{ hat einen Lösungsvektor } \overrightarrow{x} \text{ vom Gewicht } k$$
$$\Leftrightarrow$$
$$F \text{ hat eine erfüllende Belegung vom Gewicht } k.$$

Diese Reduktion erfüllt also die Forderungen von Definition 6.1, insbesondere, da sie auch in Zeit $|F|^{O(1)}$ durchgeführt werden kann.

2) Wir betrachten nun die schon bekannten (vgl. Abschnitt 2.2.1) parametrisierten Versionen der Graphprobleme Vertex Cover, Independent Set und Clique. Es gelten folgende Beziehungen:

i) Ein Graph $G = (V, E)$ hat eine Independent Set I der Größe k, genau dann wenn die Menge $V - I$ ein Vertex Cover der Größe $n - k$ ist (wobei $n := |V|$).

ii) Ein Graph $G = (V, E)$ hat ein Independent Set der Größe k, genau dann wenn der Komplementgraph $\overline{G} = (V, \overline{E})$ (d.i. der Graph, bei welchem eine Kante $\{u, v\}$ existiert, genau dann wenn $\{u, v\}$ keine Kante von G ist) eine Clique (d.h. einen vollständigen Teilgraph) der Größe k besitzt.

Die zu i) gehörige Reduktion ist keine parametrisierte Reduktion, da der „alte“ Parameter k durch den „neuen“ $k' = n - k$ ersetzt wird, welcher also im Widerspruch zu Definition 6.1 auch von der Eingabe selbst und nicht nur vom Parameter k abhängt.

Hingegen erfüllt ii) die Bedingungen von Definition 6.1, es handelt sich also um eine parametrisierte Reduktion. Clique und Independent Set sind damit auch im parametrisierten Sinne „gleich schwer".

Bemerkung 6.3.

- Es ist eher die Ausnahme als die Regel, daß „klassische Reduktionen" auch als parametrisierte Reduktionen taugen; letztere erfordern in der Regel größere Sorgfalt und mehr technischen Aufwand.
- Sollte sich „erweisen", daß irgendein parametrisiertes Problem nicht in FPT liegt, so „darf" es keine parametrisierte Reduktion von diesem Problem auf z.B. Vertex Cover oder irgendein anderes Problem in FPT geben.

6.2 Parametrisierte Komplexitätsklassen

Die NP-Vollständigkeit des Erfüllbarkeitsproblems für aussagenlogische Formeln in konjunktiver Normalform (Sat) wurde dadurch gezeigt, daß quasi die Berechnung einer nichtdeterministischen Turingmaschine mit polynomieller Laufzeit durch eine aussagenlogische Formel beschrieben wurde. Der Glauben an die Hartnäckigkeit von NP-vollständigen Problemen wie Sat, Vertex Cover oder Clique beruht insbesondere auch darauf, daß es eben als recht unwahrscheinlich gilt, daß man eine nichtdeterministischen Turingmaschine mit polynomieller Laufzeit durch eine deterministische Turingmaschine mit polynomieller Laufzeit simulieren kann. Wir haben demgemäß folgendes, einfaches NP-vollständiges Problem:

Turing Machine Acceptance:

Gegeben: Eine nichtdeterministische Turingmaschine M, ein Eingabewort x für M und eine unäre (d.h. über einem einelementigen Alphabet) kodierte Zahl m.
Frage: Besitzt M auf Eingabe x einen akzeptierenden Berechnungspfad der Länge $\leq m$?

In der „parametrisierten Welt" gibt es nun folgendes Analogon:

Short Turing Machine Acceptance:

Gegeben: Eine nichtdeterministische Turingmaschine M, ein Eingabewort x für M und ein Parameter $k \in \mathbb{N}$.
Frage: Besitzt M auf Eingabe x einen akzeptierenden Berechnungspfad der Länge $\leq k$?

Ähnlich wie sich philosophisch über die Hartnäckigkeit von Turing Machine Acceptance diskutieren läßt, ist das von einem „parametrisierten Blickwinkel" aus auch für Short Turing Machine Acceptance möglich. Es besteht wenig Hoffnung, daß Short Turing Machine Acceptance einen effizienten parametrisierten Algorithmus besitzt. D.h., daß dieses Problem wohl nicht in *FPT* liegt. (Versuche, einen Algorithmus mit Laufzeit $f(k) \cdot n^{O(1)}$ anzugeben! Gelänge dies, so wäre man reif für eine Sensation.)

Demgemäß taugt Short Turing Machine Acceptance als Kernproblem zur Betrachtung einer Klasse „harter" parametrisierter Probleme (mithilfe des Reduktionskonzepts). Leider ist das Studium „parametrisierter Hartnäckigkeit" technisch komplizierter als das vergleichbare Konzept in der klassischen Komplexitätstheorie. Demzufolge sind die zugehörigen Komplexitätsklassen auch aufwendiger zu definieren.

Der Grundgedanke bei der Einführung parametrisierter Komplexitätsklassen „überhalb" von *FPT* ist die Klassifikation von Problemen gemäß ihrer „logischen Tiefe". Im Gegensatz zur klassischen Komplexitätstheorie werden diese Komplexitätsklassen also nicht über entsprechende Maschinenmodelle definiert. Vielmehr benutzen wir folgendes Konzept:

Definition 6.4.
1. Ein *Boolesches Schaltnetz* C ist ein gerichter, azyklischer Graph, bei dem die Knoten als *Gatter* bezeichnet werden. Außerdem wird jedem Gatter eine aussagenlogische Funktion (Und, Oder, Negation) oder eine Eingabevariable zugeordnet. Schließlich gibt es noch ein ausgezeichnetes Gatter, das die Ausgabe des Schaltnetzes bestimmt.

2. *Kleine Gatter* von C sind Gatter Und, Oder, Negation, wobei der Eingangsgrad durch 2 beschränkt ist. *Große Gatter* von C sind Gatter Und, Oder, Negation mit Eingangsgrad größer als 2.[6]

3. Die *Tiefe* von C ist die maximale Anzahl von Gattern auf einem Eingabe-Ausgabe-Pfad. Die *Weft*[7] von C ist die maximale Anzahl von großen Gattern auf einem Eingabe-Ausgabe-Pfad in C.

Beispiel 6.5. Abbildung 7 zeigt ein Weft 2, Tiefe 4 Schaltnetz.

Definition 6.6. Sei $\mathcal{F} = \{C_1, C_2, C_3, \dots\}$ eine Familie von Schaltnetzen, wobei C_i genau i Eingabevariablen besitze. Wir definieren

$$L_{\mathcal{F}} := \{ < C_i, k > \mid C_i \text{ hat eine erfüllende Belegung mit Gewicht } k \}.$$

$L_{\mathcal{F}(t,h)}$ ist dann $L_{\mathcal{F}}$ eingeschränkt auf Weft t und Tiefe h Schaltnetze.

[6] Die Festlegung auf die Konstante 2 als Trennwert zwischen kleinen und großen Gattern ist hier rein willkürlich. Es hätte eine beliebige andere Konstante ≥ 2 sein können. Dies kommt z.B. auch später in der Definition 6.10 zum Tragen.

[7] Von Englisch „Schußfaden" (bei einem Webstuhl).

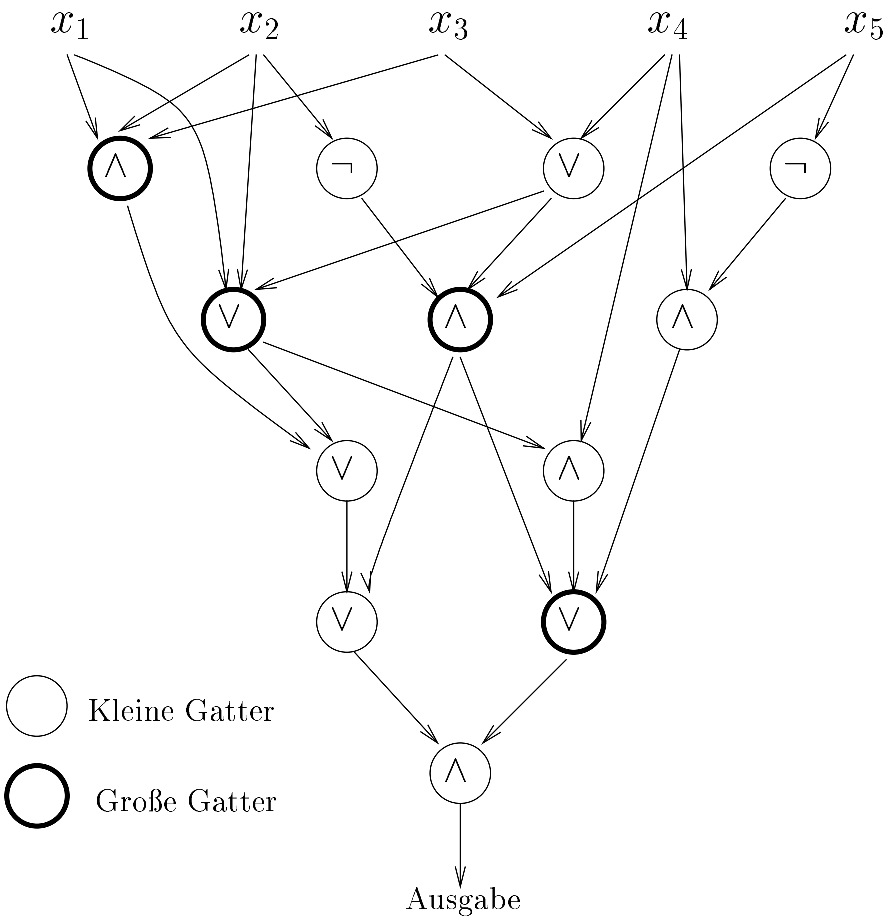


Abbildung 7: Schaltnetz der Tiefe 5 und Weft 2.

Beispiel 6.7. Weighted CNF Sat gehört zu $L_{\mathcal{F}(2,2)}$; Weighted 2CNF Sat gehört zu $L_{\mathcal{F}(1,2)}$.

Definition 6.8. Ein parametrisiertes Problem L gehört zur Komplexitätsklasse $W[t]$ genau dann, wenn es mittels einer parametrisierten Reduktion auf die Sprache einer Schaltnetzfamilie $L_{\mathcal{F}(t,h)}$ zurückgeführt werden kann, so daß die Tiefe h konstant ist.

Beispiel 6.9. 1. Independent Set und Clique sind in $W[1]$:
Beobachtung:

Ein Graph $G = (V, E)$ mit $V = \{1, 2, \dots\}$ hat eine Independent Set der Größe k.

$\Leftrightarrow$

Nachfolgende Formel hat eine erfüllende Belegung vom Gewicht k:

$$\bigwedge_{\{i,j\}\in E} (\overline{x}_i \vee \overline{x}_j). \tag{3}$$

Beachte, daß eine Independent Set genau die Eigenschaft hat, daß von je zwei benachbarten Knoten $i, j \in V$ (d.h. $\{i,j\} \in E$) höchstens einer in der Independent Set liegen kann. Dies entspricht gerade der aussagenlogischen Formel $\overline{x_i \wedge x_j} = \overline{x_i} \vee \overline{x_j}$.

Leicht zu überprüfen ist, daß die Abbildung von Independent Set auf das beschriebene Erfüllbarkeitsproblem mit einer parametrisierten Reduktion gemäß Definition 6.1 geschehen kann. Da obige Formel (3) in 2CNF ist, somit insbesondere auch durch ein Weft 1, Tiefe 2 Schaltnetz beschrieben werden kann (d.h. also auf $L_{\mathcal{F}(1,2)}$ zurückgeführt werden kann), folgt mit Definition 6.8, daß Independent Set in $W[1]$ ist.

Mit der parametrisierten Reduktion aus Beispiel 6.2.2.(ii) (Übergang auf den Komplementgraph) folgt auch, daß Clique in $W[1]$ liegt.

2. Dominating Set ist in $W[2]$:
Beobachtung:

Ein Graph $G = (V, E)$ mit $V = \{1, 2, \dots\}$ hat eine „Dominating Set" der Größe k.

$\Leftrightarrow$

Nachfolgende Formel hat eine erfüllende Belegung vom Gewicht k (dabei bezeichne $N(i)$ die Menge der Nachbarknoten des Knotens i):

$$\bigwedge_{i\in V} \quad \bigvee_{N(i)=\{i_1,\dots,i_j\}} \big(x_i, x_{i_1}, \dots, x_{i_j}\big). \tag{4}$$

Dies zeigt, daß sich Dominating Set durch eine parametrisierte Reduktion auf das in (4) beschriebene, gewichtete Erfüllbarkeitsproblem zurückführen läßt. Dieses wiederum ist durch Schaltnetze der Familie $L_{\mathcal{F}(2,2)}$ beschreibbar, sprich Dominating Set liegt in $W[2]$.

Beachte, daß es — im Gegensatz zu Formel (3) — bei der Formel (4) zweier aufeinanderfolgender Gatter mit unbeschränktem Eingangsgrad bedurfte. In diesem Sinne haben also Clique und Dominating Set verschiedene logische Tiefe. Gemäß nachfolgender Theorie käme es einer Sensation gleich, könnte man Dominating Set als ein Weighted qCNF Sat Problem (mit konstantem q) ausdrücken.

6.3 Vollständige Probleme und die W-Hierarchie

Zur Definition von $W[t]$ betrachteten wir folgende Probleme:

Weighted Weft t Depth h Circuit Satisfiability (WCS(t,h)):

Gegeben: Ein Schaltnetz C mit Weft t und Tiefe h und ein $k \in \mathbb{N}$.
Frage: Hat C eine erfüllende Belegung mit Gewicht k?

Weighted 3CNF Sat ist z.B. ein Spezialfall von WCS(1,2). Uns interessiert nachfolgend unter anderem die Frage, wie groß die Konstante h in der Definition von $W[t]$ und speziell $W[1]$ sein muß.

Definition 6.10. Unter $W[1, s]$ verstehen wir die Teilmenge von $W[1]$, welche mittels einer parametrisierten Reduktion auf die Sprache $L_{\mathcal{F}(s)}$ der Familie $\mathcal{F}(s)$ von s-normalisierten Schaltnetzen der Weft 1 und Tiefe 2 reduziert werden können. Dabei bedeutet *s-normalisiert*, daß sich in Tiefe 1 nur ODER-Gatter mit Eingangsgrad höchstens s befinden und in Tiefe 2 ein einziges, großes UND-Gatter ist.

Mit großem technischen Aufwand läßt sich folgendes Zwischenergebnis beweisen. Wir benötigen zur Formulierung des Ergebnisses noch den Begriff der „Antimonotonie“:

Definition 6.11. *Antimonotone* $W[1, s]$ ist die Einschränkung von $W[1, s]$ auf Schaltnetze, bei welchen alle Eingabevariablen negiert in das Schaltnetz gegeben werden, im eigentlichen Schaltnetz dann aber keine Negationsgatter mehr auftauchen.

Damit erhalten wir das angekündigte Resultat (vgl. [12, Lemma 10.5, Theorem 10.6]):

Mitteilung 6.12. *Es gelten folgende Aussagen:*

1. $W[1] = \bigcup_{s=1}^{\infty} W[1, s]$.
2. $W[1, s] =$ *Antimonontone* $W[1, s]$ *für* $s \geq 2$.

Mithilfe obiger Mitteilung läßt sich zeigen, daß die Schaltnetze, welche wir zur Beschreibung von $W[1]$ tatsächlich benötigen, von sehr spezieller Gestalt sein können (vgl.[12, Theorem 10.7.]). Dieses Ergebnis wird zur Beschreibung $W[1]$-vollständiger Probleme von großem Nutzen sein.

Satz 6.13. *Es gilt* $W[1] = W[1, 2]$.

Beweis.
Die Inklusion „$\supseteq$“ ist nach Definition 6.10 klar.

Für die umgekehrte Inklusion genügt dank Mitteilung 6.12 der Nachweis der Inklusion

$$\textit{Antimonotone}\, W[1,s] \subseteq W[1,2], \quad \text{für alle } s \geq 2.$$

Sei also ein Antimonotone $W[1,s]$-Schaltnetz C gegeben. Wir werden hierzu einen Ausdruck C' in 2CNF konstruieren, so daß folgendes gilt:

$$\begin{gathered} C \text{ hat eine erfüllende Belegung vom Gewicht } k \\ \Leftrightarrow \\ C' \text{ hat eine erfüllende Belegung vom Gewicht } k' := k2^k + \textstyle\sum_{i=2}^{s} \binom{k}{i}. \end{gathered}$$

Zunächst zur Konstruktion von C' aus C: Seien $x_1, \ldots, x_n$ die Eingabevariablen von C. Wir betrachten *alle* Teilmengen der Größen 2 bis s der Eingabevariablenmenge. Bezeichne diese Teilmengen mit $A_1, \ldots, A_p$. Die Eingaben eines ODER-Gatters von C entsprechen also genau einem $A_i, 1 \leq i \leq p$. Zu jedem A_i korrespondiere in C' eine neue Variable v_i. Falls v_i den Wert „wahr“ hat, so bedeutet dies also, daß *alle* Variablen, deren negierte Werte Eingänge des zu A_i gehörenden ODER-Gastters sind, ebenso „wahr“ sind. Deshalb wird in der Konstruktion von C' das zu A_i gehörende ODER-Gatter durch $\overline{v}_i$ ersetzt. Desweiteren werden für jede Eingabevariable x_j genau 2^k neue Eingabevariablen („Kopien“) $x_{j,0}, \ldots, x_{j,2^k-1}$ eingeführt.

Die Konstruktion von C' aus C sieht nun wie folgt aus:
Zunächst werden alle ODER-Gatter durch die entsprechenden Negationen von v_i's ersetzt. Desweiteren werden folgende weiteren Bestandteile jeweils per UND-Verknüpfung angehängt (diese dienen dazu, notwendige Nebenbedingungen und die Korrektheit der Behauptung zu garantieren):

a) $x_{j,r} \Rightarrow x_{j,r+1(\mathrm{mod} 2^k)}$ für $j = 1, \ldots, n$ und $r = 0, \ldots, 2^k - 1$.

b) $v_{i'} \Rightarrow v_i$ falls $A_i \subseteq A_{i'}$.

c) $v_i \Rightarrow x_{j,0}$ falls $x_j \in A_i$.

Informell ausgedrückt implizieren diese Bedingungen, daß eine Belegung von C' nur dann erfüllend sein kann, wenn folgendes gilt: Ist ein $v_{i'}$ „wahr“, so müssen auch alle v_i, welche zu Teilmengen $A_i \subseteq A_{i'}$ gehören „wahr“ sein (Bed. b)). Darüber hinaus verlangen Bedingung c) und a), daß in diesem Fall auch alle „ Kopien“ $x_{j,0}, \ldots, x_{j,2^k-1}$ von Variablen x_j, welche in diesen Teilmengen $A_i \subseteq A_{i'}$ auftauchen, „wahr“ sein müssen.

Beachte, daß die obigen Implikationen jeweils durch eine zweistellige ODER-Verknüpfung ausgedrückt werden können. Dies beschließt die Konstruktion von C'. Die Größe von C' im Vergleich zu C wächst größenordnungsmäßig nur um den Faktor 2^k, bzw. additiv um $\sum_{i=2}^{s} \binom{n}{i} = O(n^s)$ für konstantes s. Die Konstruktion ist daher durch eine parametrisierte Reduktion zu bewältigen.

Zur Korrektheit der Konstruktion und damit der Behauptung:

i) Annahme: C hat eine erfüllende Belegung vom Gewicht k, die genau die Variablen $x_{j_1}, \ldots, x_{j_k}$ mit „wahr" belegt. Dann setzte die $2^k \cdot k$ Variablen $x_{j_1,0}, \ldots, x_{j_1,2^k-1}, \ldots, x_{j_k,0}, \ldots, x_{j_k,2^k-1}$ auf „wahr" und entsprechend alle Variablen v_i, die zu einem der A_i gehören, welche eine Teilmenge der Variablen in $x_{j_1}, \ldots, x_{j_k}$ bilden. Davon gibt es $\sum_{i=2}^{s} \binom{k}{i}$ viele. Nach Konstruktion von C' gibt das eine erfüllende Belegung vom Gewicht $k2^k + \sum_{i=2}^{s} \binom{k}{i} = k'$.

ii) Umkehrannahme: C' hat eine erfüllende Belegung vom Gewicht $k' = k2^k + \sum_{i=2}^{s} \binom{k}{i}$. Bedingung a) besagt, daß für jedes j gilt:

$$x_{j,r} = \text{TRUE für ein } r \in \{1, \ldots, 2^k - 1\} \quad \Rightarrow \quad x_{j,l} = \text{TRUE für alle } l \in \{1, \ldots, 2^k - 1\} \quad .$$

Diese Eigenschaft und die Tatsache, daß $2^k > \sum_{i=2}^{s} \binom{k}{i}$ garantieren, daß für eine erfüllende Belegung von C' mit Gewicht k' *genau* k „Kopiemengen" (jede der Größe 2^k) auf „wahr" gesetzt sind. Diese entsprechen einer Belegung von C mit Gewicht k (man identifiziere wiederum jede der „Kopiemengen" $x_{j,0}, \ldots, x_{j,2^k-1}$ aus C' mit der Variablen x_j in C). Nach Konstruktion ist es leicht zu sehen, daß die so erhaltene Belegung von C erfüllend ist. Insbesondere stellen dabei die Bedingungen b) und c) sicher, daß die Belegung der Variablen v_i „kompatibel" mit den Belegungen der Kopiemengen sind.

□

Mithilfe von Satz 6.13 lassen sich nun relativ einfach Vollständigkeitsbeweise für $W[1]$ führen:

Satz 6.14. *Folgende Probleme sind $W[1]$-vollständig:*

- *Independent Set.*
- *Clique.*

- *Weighted qCNF Sat für konstantes* $q \geq 2$.
- *Short Turing Machine Acceptance.*

Beweisskizze. Leicht sieht man (vgl. das Beispiel 6.9), daß Independent Set und Clique in $W[1]$ sind. Es verbleibt also die „Härte“ nachzuweisen. Mit Satz 6.13 und Mitteilung 6.12 müssen wir hierzu zeigen, daß wir das gewichtete Erfüllbarkeitsproblem für antimonotone 2CNF-Formeln mittels einer parametrisierten Reduktion auf z.B. Independent Set (der Beweis für Clique läuft analog) zurückführen können. Sei dazu F eine antimonotone Formel in 2CNF. Wir konstruieren einen Graph G_F, bei welchem jeder Variablen in F ein Knoten in G_F und jeder Klausel eine Kante entspricht. Dann ist leicht zu sehen (man mache sich dies an einem einfachen Beispiel klar!):

$$\begin{gathered} F \text{ hat eine erfüllende Belegung vom Gewicht } k. \\ \Leftrightarrow \\ G_F \text{ hat eine Independent Set der Größe } k. \end{gathered}$$

Die Vollständigkeit von Weighted qCNF Sat ergibt sich direkt aus der Konstruktion in Satz 6.13.

Für die Vollständigkeit von Short Turing Machine Acceptance verweisen wir auf [12, S.245/250]. Die Idee besteht darin, Clique auf Short Turing Machine Acceptance (parametrisiert) zu reduzieren. Wichtig hierbei ist, daß die zu konstruierende TM ein unbeschränktes Eingabealphabet haben darf. Short Turing Machine Acceptance mit beschränktem (d.h. konstant großem) Eingabealphabet liegt in FPT. □

Abschließend wollen wir darauf hinweisen, daß es wohl Probleme gibt, welche zwischen P und NP liegen (also nicht NP-vollständig sind, aber auch nicht in P liegen), aber dennoch $W[1]$-vollständig sind. Dies gilt etwa für die Berechnung der sog. *Vapnik-Chervonenkis-Dimension*, einem Problem aus der Lerntheorie. Dies soll zeigen, daß die klassische Komplexitätstheorie und die parametrisierte Komplexitätstheorie in gewisser Hinsicht „orthogonal“ zueinander liegen: Im klassischen Sinne ist Vertex Cover „schwerer“ als die Bestimmung der Vapnik-Chervonenkis-Dimension (da Vertex Cover NP-vollständig ist), im parametrisierten Sinn jedoch „leichter“ (da in FPT).

Abschließend soll noch kurz die W-Hierarchie vorgestellt werden, welche parametrisierte Probleme — wie bereits erwähnt — gemäß ihrer logischen Tiefe klassifiziert:

Definition 6.15. Die *W-Hierarchie* ist die Vereinigung aller parametrisierten Komplexitätsklassen $W[t]$, $t \geq 1$, zusammen mit den zwei zusätzlichen

Klassen $W[\mathrm{Sat}]$ und $W[P]$, wobei diese analog zu den $W[t]$-Klassen definiert sind, nur daß bei $W[P]$ keine Beschränkung der Schaltnetztiefe besteht und bei $W[\mathrm{Sat}]$ aussagenlogische Formeln polynomieller Größe (anstelle von Schaltnetzen polynomieller Größe wie bei $W[P]$) verwendet werden. Somit haben wir

$$\mathrm{FPT} \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq W[\mathrm{Sat}] \subseteq W[P].$$

Es wird vermutet, daß obige Inklusionen alle echt sind. Es läßt sich im übrigen einfach zeigen, daß mit $P = NP$ bereits $W[P] = FPT$ gelten würde. Die Gültigkeit der Umkehrung dieser Aussage ist nicht bekannt.

Zur Angabe vollständiger Probleme für die W-Hierarchie brauchen wir noch:

Definition 6.16. Eine aussagenlogische Formel heißt t-normalisiert, falls sie von der Form „Produkt-von-Summen-von-Produkten-von-...“ von Literalen ist mit $t-1$ Alternierungen.

Beispiel 6.17. CNF ist 2-normalisiert.

Mitteilung 6.18. *Weighted t-normalized Sat ist $W[t]$-vollständig für $t \geq 2$.*

Mitteilung 6.19. *Dominating Set ist $W[2]$-vollständig.*

Beispiel 6.20. Abschließend wollen wir ein Beispiel für die verschiedenen Parametrisierungsmöglichkeiten und damit verbunden die verschiedenen parametrisierten Komplexitäten eines Problems geben.

Longest Common Subsequence Problem (LCS)

Gegeben: k Zeichenketten $X_1, \ldots, X_k$ über einem Alphabet Σ und $m \in \mathbb{N}$.
Frage: Existiert $X \in \Sigma^*$ mit $|X| \geq m$, so daß X Teilsequenz(Beachte, daß hier nicht Teilzeichenkette gemeint ist.) aller X_i ist?

LCS spielt eine wichtige Rolle in der algorithmischen Biologie. Es läßt sich auf verschiedene Weisen „parametrisieren“:

LCS1: Nur k als Parameter.

LCS2: Nur m als Parameter.

LCS3: k und m als Parameter.

Der gegenwärtige Kenntnisstand über die parametrisierte Komplexität dieser Varianten läßt sich folgender Tabelle entnehmen:

Problem	Parameter	$\lvert\Sigma\rvert$ unbeschränkt	$\lvert\Sigma\rvert$ beschränkt
LCS1	k	$W[t]$-hart für alle $t \geq 1$	ungelöst/offen
LCS2	m	$W[2]$-hart	FPT
LCS3	k, m	$W[1]$-vollständig	FPT

Es ist also immer sorgfältig zu untersuchen, welche Parametrisierung für das gegebene Problem (anwendungsbedingt) sinnvoll ist.

6.4 Strukturelle Ergebnisse

In diesem Abschnitt sollen noch kurz Indizien aufgezeigt werden, die für die „Echtheit“ der W-Hierarchie und insbesondere die Hartnäckigkeit von $W[1]$-harten Problemen sprechen. Die fundamentale Hypothese der parametrisierten Komplexitätstheorie ist die Ungleichheit

$$W[1] \neq FPT.$$

Eine weitere Hypothese ist beispielsweise

$$W[1] \neq W[2].$$

Da es beim gegenwärtigen Stand der Forschung als recht hoffnungslos gilt, eine Aussage wie $W[1] \neq FPT$ zeigen zu können, geht man über zu „relativen Aussagen“. Würde also z.B. die Annahme $W[1] = FPT$ eine Aussage (in der klassischen Komplexitätstheorie) implizieren, welche als sehr unwahrscheinlich oder überraschend gilt? In diese Richtung stoßen die nachfolgend vorgestellten Sätze, die ohne Beweis nur als Mitteilung gegeben werden.

Leider ist *keine* derart starke (relative) Aussage wie etwa

$$W[P] = FPT \Rightarrow P = NP$$

bekannt. Deshalb müssen wir zu „schwächeren Aussagen“ übergehen, wozu das Konzept des „beschränkten Nichtdeterminismus“ benötigt wird.

Definition 6.21.
1. $NP[f(n)]$ bezeichne die Klasse von Entscheidungsproblemen, die in polynomieller Zeit von einer nichtdeterministischen Turingmaschine gelöst werden können, welche nur $f(n)$ viele nichtdeterministische Schritte erlaubt.
2. $SUBEXPTIME\,[f(n)]$ bezeichne die Klasse von Sprachen, die von einer deterministischen Turingmaschine in Zeit $n^{O(1)} \cdot 2^{g(n)}$ für ein g in $o(f(n))$ akzeptiert werden können.

Mitteilung 6.22. *Folgende Aussagen sind äquivalent:*

(i) $W[P]$ =*FPT*.

(ii) *Es gilt* $NP[f(n)] \subseteq \text{SUBEXPTIME}\ [f(n)]$ *für alle in Polynomzeit berechenbaren Funktionen* $f(n)$ *mit* $f(n) \geq \log n$.

Beachte, daß obige Aussage (ii) anschaulich bedeutet, daß für jedes $NP[f(n)]$-Problem ein deterministischer Algorithmus gefunden werden kann, welcher mit weniger Schritten auskommt, als der naive Ansatz, welcher alle exponentiell vielen Möglichkeiten durchprobiert. Dies gilt in der klassischen Komplexitätstheorie als sehr unwahrscheinlich.

Mitteilung 6.23. $W[1]$ =*FPT* $\Rightarrow$ *3CNF Sat kann in Zeit* $2^{o(n)}$ *gelöst werden.*

Auch diese Folgerung gilt in der klassischen Komplexitätstheorie als sehr unwahrscheinlich. Der bislang beste Algorithmus für 3CNF Sat hat Laufzeit $\approx 1.5^n = 2^{O(n)} \neq 2^{o(n)}$ (siehe [17, 26]).

7 Blick zurück und nach vorn

Das Studium parametrisierter Algorithmen und ihrer Komplexität ist ein junges und vielversprechendes Forschungsbgebiet mit unstrittiger praktischer Relevanz. Neue Techniken (Reduktion auf den Problemkern, tiefenbeschränkter Suchbaum, Farbkodierungen etc.) sind die Grundlage für effiziente parametrisierte Algorithmen. Ähnlich wie in der klassischen Komplexitätstheorie P und NP, so stehen sich in der parametrisierten Komplexitätstheorie FPT und $W[1]$(-Härte) als gutartige und bösartige Probleme gegenüber. Leider ist bislang noch nicht der Beleg erbracht, daß die Mehrzahl der Probleme in FPT wirklich effiziente, praxistaugliche Algorithmen besitzen. Dies muß Gegenstand zukünftiger Forschung sein.

Zusammenfassend lassen sich folgende Themen als Höhepunkte des behandelten Stoffs betrachten.

- Vertex Cover und seine effizienten parametrisierten Algorithmen.
- parametrisierte Algorithmen im Vergleich mit anderen Ansätzen zur Handhabung kombinatorisch schwieriger Probleme.
- Fixed parameter tractable Probleme.
- Reduktion auf den Problemkern.
- Suchbäume beschränkter Tiefe.
- Farbkodierungen und Hashing.
- Approximation und parametrisierte Algorithmen.
- parametrisierte Reduktion.
- W-Hierarchie und vollständige Probleme.

Die Zukunft des Gebiets der parametrisierten Algorithmen hängt eng mit ihrer Bewährung in der Praxis zusammen. Bislang wurden noch relativ wenige der Algorithmen implementiert. Außerdem ist von entscheidender Bedeutung, ob FPT wirklich auch als Konzept für *effiziente* parametrisierte Algorithmen tragfähig ist. Letzlich entscheidet sich das Schicksal parametrisierter Algorithmen wohl auch im direkten Vergleich mit in der Praxis eingesetzten heuristischen Verfahren und den Bezügen zwischen beiden Ansätzen, ggf. auch ihrer Kombinierbarkeit. Viel ist noch zu tun — auch im Rahmen von Studien- und Diplomarbeiten, die sowohl theoretisch als auch praktisch ausgerichtet sein können.

Literatur

[1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33d IEEE Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[3] R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[4] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Disc. Math.*, 25:27–46, 1985.

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

[6] J. F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22(3):560–572, 1993.

[7] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *Journal of Computer and System Sciences*, 54:465–474, 1997.

[8] J. Chen, I. Kanj, and W. Jia. Vertex cover: Further observations and further improvements. To appear at *25th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'99),* Ascona, Switzerland, June 1999.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[10] B. Courcelle. The monadic second order theory of graphs I: Recognisable sets of finite graphs. *Information and Computation*, 85:12–75, 1990.

[11] P. Crescenzi and V. Kann. How to find the best approximation results—a follow-up to Garey and Johnson. *ACM SIGACT News*, 29(4):90–97, 1998.

[12] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, 1999.

[13] R. G. Downey, M. R. Fellows, and U. Stege. Parameterized complexity: A framework for systematically confronting computational intractability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 49, 1999.

[14] R. C. Evans. Testing repairable RAMs and mostly good memories. In *Proceedings of the IEEE Int. Test Conf.*, pages 49–55, 1981.

[15] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Freeman, San Francisco, 1979.

[16] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 1–10, 1997.

[17] O. Kullmann. New methods for 3-SAT decison and worst-case analysis. *Theoretical Computer Science*, 223:1–72, 1999.

[18] K. Mehlhorn. *Graph algorithms and NP-completeness.* Heidelberg: Springer, 1984.

[19] B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.

[20] R. Niedermeier. Some prospects for efficent fixed parameter algorithms (invited paper). In B. Rovan, editor, *Proceedings of the 25th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, number 1521 in Lecture Notes in Computer Science, pages 168–185. Springer-Verlag, 1998.

[21] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. Technical Report TUM-I9913, Institut für Informatik, Technische Universität München,Fed. Rep. of Germany, June 1999. Submitted to *Information Processing Letters*.

[22] R. Niedermeier and P. Rossmanith. Upper bounds for Vertex Cover further improved. In C. Meinel and S. Tison, editors, *Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science*, number 1563 in Lecture Notes in Computer Science, pages 561–570. Springer-Verlag, 1999.

[23] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[24] V. Raman. Parameterized complexity. In *Proceedings of the 7th National Seminar on Theoretical Computer Science (Chennai, India)*, pages I–1–I–18, June 1997.

[25] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7:425–440, 1986.

[26] I. Schiermeyer. Pure literal look ahead: An $O(1.497^n)$ 3-Satisfiability algorithm. Technical Report 96-230, Universität Köln, 1996.

[27] U. Stege and M. Fellows. An improved fixed-parameter-tractable algorithm for vertex cover. Technical Report 318, Department of Computer Science, ETH Zürich, April 1999.

[28] M. Thorup. Structured programs have small tree-width and good register allocation. In R. Möhring, editor, *Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science*, number 1335 in Lecture Notes in Computer Science, pages 318–332. Springer-Verlag, 1997.

Index