Klassen von Approximationsproblemen

Matthias Ruhl

Diplomarbeit Informatik

Fakultät für angewandte Wissenschaften
Albert-Ludwigs-Universität Freiburg
Prof. Dr. Jörg Flum

April 1998

Inhaltsverzeichnis

Vorwort

Diese Diplomarbeit beschäftigt sich mit der Frage, ob die Komplexitätsklassen **FPTAS** und **PTAS** eine syntaktische Charakterisierung haben. Lösen werden wir diese Frage nicht, wohl aber sehen, wie nah (oder fern) die Forschung momentan ihrer Beantwortung ist.

Ich danke Herrn Professor Jörg Flum für die Ermöglichung und Betreuung dieser Arbeit, und Martin Grohe für seine hilfreichen Kommentare.

Freiburg, im April 1998 Matthias Ruhl

Vorbemerkungen

Diese Arbeit führt Begriffe, die man im Rahmen eines normalen Informatik-Studiums kennengelernt haben sollte, nicht explizit ein. Grundlagen der theoretischen Informatik findet man in [Schöning97], eine Einführung in die Komplexitätstheorie in [Papadimitriou94], und Grundlagen der Logik in [EFT96].

In dieser Arbeit gilt

$$\mathbb{N} = \{\ 0, 1, 2, 3, \dots\ \}$$

Erklärung

Ich versichere hiermit, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Freiburg, 16. April 1998

Kapitel 1

Einleitung

1.1 Deskriptive Komplexitätstheorie

Die Komplexitätstheorie beschäftigt sich mit der Frage, wie aufwendig es ist, Probleme mit einem Computer zu lösen. Sie versucht dabei, gegebene Probleme anhand ihrer Schwierigkeit in Komplexitätsklassen wie **P** oder **NP** einzuteilen.

Die *deskriptive Komplexitätstheorie* verfolgt das gleiche Ziel, benutzt jedoch andere Methoden. Sie untersucht den Zusammenhang zwischen der Beschreibung eines Problems und seiner Komplexität. Kann man allein an der Beschreibung eines Problems erkennen, wie schwer es ist? In vielen Fällen geht das tatsächlich.

1974 zeigte Ron Fagin, daß sich alle Probleme aus **NP** durch eine Sprache (genauer: Logik) namens Σ^1_1 beschreiben lassen [Fagin74]. Und umgekehrt entspricht jeder Ausdruck dieser Sprache einem Problem aus Σ^1_1. Ähnliche Resultate wurden in den letzten Jahren für **P** [Immerman86, Vardi82], **PSPACE** [AbVi89], **LOGSPACE** [Immerman87], und andere Klassen gewonnen.

Wir nennen einen solchen Sachverhalt eine *syntaktische Beschreibung* einer Komplexitätsklasse. Der vielleicht wichtigste Grund, um nach syntaktischen Beschreibungen von Komplexitätsklassen zu suchen, ist der folgende. Haben wir eine solche Beschreibung, so haben wir einerseits das Gefühl, die Klasse sehr gut zu verstehen. Denn wir können sehen, was typisch und untypisch für diese Klasse ist. Das hilft uns, Eigenschaften über die Klasse zu beweisen, und auch zukünftige Probleme schneller auf ihre Zugehörigkeit zu der Klasse zu testen. Und oft liefert uns die syntaktische Beschreibung einer Klasse auch gleich kanonische Lösungsalgorithmen für ihre Probleme mit.

In dieser Arbeit stellen wir Resultate zur syntaktischen Charakterisierung der zwei Komplexitätsklassen **FPTAS** und **PTAS** vor, die in den letzten Jahren gewonnen wurden. Da dies Klassen von *Optimierungsproblemen* sind, werden wir uns diesen nun zuwenden.

1.2 Optimierungsprobleme

Traditionell beschäftigte sich die Komplexitätstheorie mit *Entscheidungsproblemen*. Das sind Probleme, bei denen für jede Eingabe die Antwort entweder 'ja' oder 'nein' ist.

Leider (oder zum Glück) ist das wirkliche Leben nicht so einfach. Für die meisten Problem gibt es nicht eine, sondern viele Lösungen. So gibt es viele Möglichkeiten, von Freiburg nach München zu fahren.

Das soll jedoch nicht heißen, das alle diese Lösungen gleich *gut* sind. Meistens wollen wir einen gewissen Wert maximieren oder minimieren, wie z.B. die Kosten oder die Fahrtdauer.

Deswegen wurden die 90er Jahre zum Jahrzehnt der Optimierungsprobleme, und auch wir beschäftigen uns in dieser Arbeit damit. Die meisten interessanten Optimierungsprobleme liegen in der Klasse **NPO**, die wir nun formal definieren.

Definition 1.1 (NP-Optimierungsproblem, NPO)
Ein **NP-Optimierungsproblem** $\mathcal{O}$ *ist ein Quadrupel* $\mathcal{O} = (I, F, cost, opt)$, *dabei sind*

- I *die Klasse der möglichen Eingaben (z.B. Graphen, Funktionen über einem endlichen Universum, usw.). Die Zugehörigkeit zu* I *ist in polynomieller Zeit entscheidbar.*[1]

- F *eine Funktion auf* I*, die jeder Eingabe* $\mathcal{I} \in I$ *eine Menge von* zulässigen Lösungen *zuordnet. Die Menge* $\{\ (\mathcal{I}, S) \mid \mathcal{I} \in I,\ S \in F(\mathcal{I})\ \}$ *ist in in der ersten Komponente polynomieller Zeit entscheidbar.*[2]

- $cost$ *ist eine Funktion von* $I \times F(I)$ *nach* $\mathbb{N}$*, ebenfalls in polynomieller Zeit berechenbar.*[3]

---

[1] Genauer bedeutet das, daß I als Teilmenge von $\{0,1\}^*$ kodiert werden kann, die in polynomieller Zeit entscheidbar ist. ($\{0,1\}^*$ ist die Menge aller endlichen Worte, die aus den Symbolen 0 und 1 bestehen, siehe [Schöning97].)

[2] Insbesondere sind die Elemente von $F(\mathcal{I})$ also nur polynomiell in $|\mathcal{I}|$ (der Länge der Kodierung über dem Alphabet $\{0,1\}$) groß.

[3] Uns interessieren im folgenden dabei nur Werte die $cost(\mathcal{I}, S)$ mit $S \in F(\mathcal{I})$, weshalb wir bei Problemdefinitionen auch nur diese angeben. Die anderen Werte seien dann beliebig festgelegt.

- *opt* $\in \{\min, \max\}$, *je nachdem ob das Problem ein* Minimierungs- *oder* Maximierungsproblem *ist.*

Die Klasse aller NP-Optimierungsprobleme bezeichnen wir mit **NPO**. □

Die beste Möglichkeit zu veranschaulichen, wie Probleme aus **NPO** aussehen können, ist es, sich einige anzuschauen. Dem Leser werden vermutlich einige der nun folgenden Probleme bereits bekannt sein. Umfassende Sammlungen solcher Probleme finden sich in [GaJo79, CrKa95].

1.3 Beispiele von Optimierungsproblemen

1.3.1 Travelling Salesman

Das "Travelling Salesman Problem", kurz TSP, ist vermutlich das bekannteste NP-Optimierungsproblem.

Ein Handlungsreisender soll n Städte nacheinander besuchen. Die Kosten der Reise von einer Stadt zu einer anderen sind für jedes Städtepaar gegeben. Gesucht ist die Reiseroute (also Reihenfolge der Städte), die die Gesamtreisekosten minimiert. Formal sieht das so aus:

TSP$= (C, T, cost, \min)$

C ist die Menge der Funktionen c von $\{1, \ldots, n\} \times \{1, \ldots, n\}$ nach $\mathbb{N} \cup \{\infty\}$. $c(i, j)$ gibt die Kosten der Reise von der i-ten zur j-ten Stadt an.

$T(c)$ ist die Menge aller Permutationen π auf $\{1, \ldots, n\}$. π gibt an, in welcher Reihenfolge die Städte besucht werden.

$cost(c, \pi)$ sind die Gesamtkosten der Reise bei Verbindungskosten c und Besuchsreihenfolge π, also

$$cost(c, \pi) = \sum_{i=1}^{n-1} c(\pi(i), \pi(i+1))$$

1.3.2 Maxcut

MAXCUT ist ein etwas abstrakteres Optimierungsproblem. Für einen gegebenen Graphen suchen wir eine Aufteilung der Knoten in zwei Mengen, so daß die Anzahl der Kanten zwischen den beiden Mengen maximiert wird.

MAXCUT=$(G, P, cost, \max)$:

G ist die Menge aller endlichen ungerichteten Graphen.

$P(\mathcal{G})$ ist für jeden Graph $\mathcal{G} = (V, E)$ gleich der Potenzmenge der Knoten von $\mathcal{G}$, also der Menge aller Teilmengen von V.

$cost(\mathcal{G}, T)$ ist für jeden Graphen $\mathcal{G} = (V, E)$ und jede Teilmenge der Knoten $T \subseteq V$ die Anzahl der Kanten von T nach $V - T$:

$$cost(\mathcal{G}, T) = |\ \{\ (v, w) \in E \mid v \in T,\ w \notin T\ \}\ |$$

1.3.3 Maximum Independent Set

MAXIMUM INDEPENDENT SET basiert ebenfalls auf Graphen. Wir suchen die größte Teilmenge T der Knoten eines Graphen, so daß keine zwei Elemente von T durch eine Kante verbunden sind (eine solche Menge nennt man auch *unabhängig* – daher der Name des Problems.)

MAXIMUM INDEPENDENT SET = $(G, P, cost, \max)$

G ist die Menge aller endlichen Graphen,

$P(\mathcal{G})$ ist für jeden Graph $\mathcal{G} = (V, E)$ die Menge der Teilmengen T von V, so daß keine zwei Elemente von T durch eine Kante des Graphen verbunden sind, d.h.

$$P(\mathcal{G}) = \{\ T \subseteq V \mid \forall v, w \in T : (v, w) \notin E\ \}$$

$cost(\mathcal{G}, T)\ = |\ T\ |$

1.3.4 Knapsack

KNAPSACK ist wieder ein realistisches Optimierungsproblem. Wir haben mehrere Gegenstände gegeben, die jeweils ein gewisses Gewicht und einen gewissen Wert haben. Wir wollen einige dieser Gegenstände in einen Rucksack packen, so daß wir ihn noch tragen können, der Wert dieser Gegenstände aber maximal wird.

KNAPSACK=$(F, P, cost, \max)$

F ist die Menge der Tripel (g, w, b), wobei $b \in \mathbb{N}$ und g, w Funktionen von $\{1, \dots, n\}$ nach $\mathbb{N}$ sind. (g und w beschreiben das Gewicht und den Wert der n Gegenstände, b ist das größte Gesamtgewicht, das wir noch tragen können.)

$P((g, w, b))$ ist die Menge aller Teilmengen $T \subseteq \{1, \dots, n\}$, für die gilt:

$$\sum_{i \in T} g(i) \leq b$$

$cost((g, w, b), T)$ ist der Wert der Gegenstände in der Menge T

$$cost((g, w, b), T) = \sum_{i \in T} w(i)$$

1.4 Approximative Lösungen

Wenn wir die Eingabe $\mathcal{I}$ zu einem Optimierungsproblem $\mathcal{O}$ gegeben haben, so ist es unser Ziel, eine möglichst gute Lösung für das Problem zu finden. Die Kosten der besten Lösung bezeichnen wir mit $opt_{\mathcal{O}}(\mathcal{I})$.

Definition 1.2 ($opt_{\mathcal{O}}(\mathcal{I})$)
Sei $\mathcal{O} = (I, F, cost, opt)$ ein NP-Optimierungsproblem und $\mathcal{I} \in I$. Dann setzen wir

$$opt_{\mathcal{O}}(\mathcal{I}) := \begin{cases} \max\limits_{S \in F(\mathcal{I})} cost(\mathcal{I}, S), & \text{falls } opt = \max, \\ \min\limits_{S \in F(\mathcal{I})} cost(\mathcal{I}, S), & \text{falls } opt = \min \end{cases}$$

□

Gilt $\mathbf{P} \neq \mathbf{NP}$, so ist es für viele Probleme aus **NPO** nicht möglich, in polynomieller Zeit (also schnell) eine optimale Lösung zu finden. Wir suchen daher nach Algorithmen, die Lösungen liefern, deren Abstand zum Optimum beschränkt ist.

Definition 1.3 (**ε-approximative Lösung**)
Sei $\mathcal{O} = (I, F, cost, opt)$ ein NP-Optimierungsproblem und $\mathcal{I} \in I$. Dann ist $S \in F(\mathcal{I})$ eine ε-approximative Lösung zu $\mathcal{I}$, wenn folgendes gilt.[4]

[4]Damit die Definition in allen Fällen Sinn macht, setzen wir $\frac{n}{0} := +\infty$, falls $n > 0$, sowie $\frac{0}{0} := 1$.

Falls opt = max*:*

$$1 - \varepsilon \leq \frac{cost(S)}{opt_{\mathcal{O}}(\mathcal{I})}$$

Falls opt = min*:*

$$1 + \varepsilon \geq \frac{cost(S)}{opt_{\mathcal{O}}(\mathcal{I})}$$

□

Das ε in dieser Definition gibt also den relativen "Fehler" an, den die betreffende Lösung gegenüber der optimalen Lösung macht, z.B. einen 50%-igen Fehler bei $\varepsilon = \frac{1}{2}$, oder einen 10%-igen bei $\varepsilon = \frac{1}{10}$.

1.5 Komplexitätsklassen

Für die Probleme aus **NPO** ist es unterschiedlich schwer, approximative Lösungen zu finden.

Bei manchen Problemen ist dies ganz einfach: für jedes $\varepsilon > 0$ kann man eine ε-approximative Lösung in polynomieller Zeit finden. Für andere Probleme ist das Finden einer approximativen Lösung genauso schwierig wie das Finden einer optimalen Lösung.

Man klassifiziert daher die Probleme aus **NPO** anhand der Schwierigkeit, approximative Lösungen zu finden. Die wichtigsten dieser Komplexitätsklassen wollen wir nun einführen.

1.5.1 APX

Definition 1.4 (PO)
PO *ist die Menge der NP-Optimierungsprobleme, für die sich in polynomieller Zeit eine optimale Lösung berechnen läßt.* □

Definition 1.5 (APX)
APX *ist die Menge der NP-Optimierungsprobleme, die sich für ein* $\varepsilon > 0$ *in polynomieller Zeit* ε*-approximativ lösen lassen.* □

Offenbar gilt trivialerweise **PO** $\subseteq$ **APX**. Ein nicht triviales Beispiel für ein Problem aus **APX** ist MAXCUT. Schon 1976 wurde ein Approximationsalgorithmus für MAXCUT entdeckt, der $\varepsilon = \frac{1}{2}$ erreichte [SaGo76].[5]

[5] Er basiert im wesentlichen auf der Tatsache, daß der Erwartungswert der Kosten einer *zufällig* gewählten Lösung schon die Hälfte des Optimums ist, und sich dieser Algorithmus auch derandomisieren läßt.

1994 entwickelten Goemans und Williamson [GoWi95] einen Algorithmus, der sogar $\varepsilon = 0,12144$ erreicht. Erstaunlicherweise konnten Arora et al. jedoch zeigen [ALMSS92], daß sich MAXCUT nicht beliebig gut approximieren läßt. Die beste bislang bekannte Schranke [Håstad97] besagt, daß es keinen polynomiellen Approximationsalgorithmus mit $\varepsilon \leq \frac{1}{17} \approx 0,0588$ geben kann. Die genaue Grenze zwischen Approximierbarkeit und Nicht-Approximierbarkeit für MAXCUT ist jedoch (noch) nicht bekannt.

1.5.2 PTAS

Definition 1.6 (PTAS)
PTAS *ist die Menge der NP-Optimierungsprobleme, für die ein Algorithmus existiert, der als zusätzliche Eingabe eine Zahl* $\varepsilon > 0$ *nimmt, und dann eine* ε*-approximative Lösung ermittelt. Hierbei muß die Laufzeit bei festem* ε *polynomiell sein.* □

Die Laufzeit kann dabei durchaus gewaltig ansteigen bei kleiner werdendem ε; typische Laufzeiten sind von der Größenordnung $O(n^{\frac{1}{\varepsilon}})$, wobei n die Größe der Eingabe ist. **PTAS** ist eine Abkürzung für **P**olymonial-**T**ime **A**ppromixation **S**cheme.

Ein Beispiel für ein Problem aus **PTAS** ist PLANAR MAXIMUM INDEPENDENT SET, das man aus dem in Abschnitt 1.3.3 definierten MAXIMUM INDEPENDENT SET erhält, wenn man als Eingabe nur planare Graphen zuläßt. Näheres dazu in Kapitel 3.

1.5.3 FPTAS

Definition 1.7 (FPTAS)
FPTAS *ist die Menge der NP-Optimierungsprobleme, für die ein Algorithmus existiert, der als zusätzliche Eingabe eine Zahl* $\varepsilon > 0$ *nimmt, und dann eine* ε*-approximative Lösung ermittelt. Hierbei muß die Laufzeit polynomiell in* n *(der Größe der Eingabe) und* $\frac{1}{\varepsilon}$ *sein (d.h.* $O(\frac{n^k}{\varepsilon^l})$ *für geeignete* $k, l \in \mathbb{N}$*).*
□

FPTAS steht für **F**ully **P**olymonial-**T**ime **A**ppromixation **S**cheme. Das klassische Beispiel hierfür ist das in Abschnitt 1.3.4 definierte KNAPSACK – mehr dazu in Kapitel 2.

1.5.4 Abhängigkeiten

Satz 1.8
Es gilt

$$\mathbf{PO} \subseteq \mathbf{FPTAS} \subseteq \mathbf{PTAS} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}$$

Gilt $\mathbf{P} \neq \mathbf{NP}$*, so sind alle Inklusionen echt.*

Beweis: Daß die Inklusionen gelten, folgt direkt aus den Definitionen. Gilt $\mathbf{P} \neq \mathbf{NP}$, so sind sie echt, denn

1. $\mathbf{PO} \neq \mathbf{FPTAS}$, da Knapsack in der Entscheidungsversion **NP**-vollständig ist, eine optimale Lösung also nicht in polynomieller Zeit berechenbar ist.

2. $\mathbf{FPTAS} \neq \mathbf{PTAS}$, da Planar Maximum Independent Set ebenfalls in der Entscheidungsversion **NP**-vollständig ist. Und die Kostenfunktion dieses Problems nimmt bei einem Graphen der Größe (Knotenanzahl) n höchstens den Wert n an. Löst man das Problem approximativ mit $\varepsilon = \frac{1}{n+1}$, so hat deshalb bereits die optimale Lösung. Läge das Problem in **FPTAS**, so existierte also ein polynomieller Algorithmus, der die optimale Lösung liefert – im Widerspruch zur **NP**-Vollständigkeit.

3. $\mathbf{PTAS} \neq \mathbf{APX}$, da Maxcut zwar in **APX** liegt, aber nicht beliebig approximierbar ist [ALMSS92].

4. $\mathbf{APX} \neq \mathbf{NPO}$, da TSP nicht approximativ in polynomieller Zeit lösbar ist (siehe etwa [Papadimitriou94], Theorem 13.4). ■

1.6 APX

Am Anfang dieses Kapitels sprachen wir über syntaktische Charakterisierungen von Komplexitätsklassen. Als Beispiel einer Klasse, bei der dies gelungen ist, betrachten wir in diesem Abschnitt die Klasse **APX**.

1989: MAXSNP

1989 führten Papadimitriou und Yannakakis die Problemklasse **MAXSNP** ein [PaYa91]. Zur Definition benutzen wir einige wenige Begriffe aus der Logik, die wir als bekannt vorraussetzen (siehe etwa [EFT96, EbFl95]).

Definition 1.9 (MAXSNP)
Seien τ *und* σ *endliche, disjunkte Mengen von Relationssymbolen, und* $\varphi(\overline{x}, \overline{S})$ $(\overline{S} \in \sigma)$ *eine quantorenfreie* $(\tau \cup \sigma)$*-Formel der Logik erster Stufe. Dann liegt folgendes Optimierungsproblem* $(I, F, cost, \max)$ *in* **MAXSNP***:*

I *ist eine Klasse endlicher* τ*-Strukturen. Die Mitgliedschaft in* I *ist polynomiell entscheidbar.*

$F(\mathcal{A})$ enthält alle möglichen Belegungen $\overline{S}$ der Relationen aus σ über dem Universum von $\mathcal{A}$

$cost(\mathcal{A}, \overline{S}) = |\ \{\ \overline{x} \mid (\mathcal{A}, \overline{S}) \models \varphi(\overline{x}, \overline{S})\}\ | \ \Box$

Zum Beispiel liegt Maxcut in **MAXSNP** mittels $\varphi(x, y) = Exy \wedge Sx \wedge \neg Sy$. Papadimitriou und Yannakakis zeigten, daß alle Probleme aus **MAXSNP** in **APX** liegen, und sich aus der Beschreibung eines Problems mittels einer Formel φ kanonisch ein Approximationsalgorithmus gewinnen läßt.

Weiterhin führten sie eine approximation-erhaltende Reduktion ein, und konnten so **MAXSNP**-vollständige Probleme definieren.

1992: PCP

1992 konnten Arora et al. [ALMSS92] beweisen, daß kein **MAXSNP**-vollständiges Probleme in **PTAS** liegt, sich also beliebig gut approximieren läßt. Hier konnte die syntaktische Charakterisierung also bereits helfen, eine ganze Reihe von Problemen auf einmal neu zu klassifizieren.

1994: $\overline{\textbf{MAXSNP}} = \textbf{APX}$

Khanna et al. [KMSV94] zeigten 1994, daß der Abschluß von **MAXSNP** unter geeigneten, die Approximierbarkeit erhaltenden, Reduktionen gleich **APX** ist. Die Probleme aus **MAXSNP** sind also in einem gewissen Sinne repräsentativ für alle Probleme aus **APX**.

1.7 Der Rest dieser Arbeit

Der erste Schritt zum Finden einer syntaktischen Charakterisierung einer Komplexitätsklasse – vielleicht überhaupt zum richtigen Verständnis einer Komplexitätsklasse – ist das Auffinden typischer Probleme für sie.

Der nächste Schritt ist die Definition einer syntaktischen Problemklasse, die die typischen Probleme der Klassen umfaßt. Diesen Schritt betrachten wir für die Komplexitätsklassen **FPTAS** (Kapitel 2) und **PTAS** (Kapitel 3) in dieser Arbeit.

Das Vorgehen ist dabei in beiden Fällen grundsätzlich das gleiche: zunächst definieren wir eine Problemklasse (**FPTAS**) oder einzelne Probleme (**PTAS**). Dann zeigen wir, daß bekannte Probleme in dieser Klasse liegen bzw. sich auf die definierten Probleme reduzieren lassen. Schließlich, und dies ist der längste Teil der Kapitel, konstruieren wir FPTAS- bzw. PTAS-Algorithmen für die definierten Probleme.

Wir beschließen die Arbeit in Kapitel 4 mit einer kurzen Diskussion der Ergebnisse.

Kapitel 2

FPTAS

Die Klasse **FPTAS** enthält die 'Harmlosesten' der NP-Optimierungsprobleme. Schließlich steigt der Rechenaufwand auch bei kleiner werdendem ε nur in vernünftigem Maße an.

In diesem Kapitel führen wir die Klasse **MAX-DP** ein, die vollständig in **FPTAS** liegt, und bekannte Probleme aus **FPTAS** enthält. Die Ergebnisse dieses Kapitels beruhen auf Resultaten, die Anders Malmström in seiner Dissertation [Malmström96a] aufgestellt hat. Unsere Ergebnisse sind etwas allgemeiner, kommen ohne den komplizierten Apparat der von Malmström benutzten meta-endlichen Modelltheorie aus, und haben deutlich einfachere Beweise.

2.1 Notation

Die Eingaben und Lösungen der in diesem Kapitel betrachteten Klasse **MAX-DP** werden Funktionen sein. Um einfacher über Funktionen sprechen zu können, brauchen wir zunächst einige Begriffe.

Definition 2.1 ($\mathcal{F}(A, B)$)
Sind A und B Mengen, so bezeichnen wir mit $\mathcal{F}(A, B)$ die Menge der Funktionen von A nach B. $\Box$

Definition 2.2 ($\sqsubseteq$, $\sqsubset$)
Sei A eine Menge und $f, g \in \mathcal{F}(A, \mathbb{N})$ Funktionen von A nach $\mathbb{N}$, dann gilt $f \sqsubseteq g$, falls f "kleiner gleich" *g ist, d.h.*

$$f \sqsubseteq g \;:\Leftrightarrow\; \forall\, a \in A \;:\; f(a) \leq g(a)$$

Analog $f \sqsubset g \;:\Leftrightarrow\; f \sqsubseteq g \wedge f \neq g$. $\Box$

Definition 2.3 ($Tr(f)$)
Sei $f \in \mathcal{F}(A, \mathbb{N})$. Dann bezeichnen wir mit $Tr(f)$ den Träger von f, *d.h.*

$$Tr(f) := \{a \in A \mid f(a) \neq 0\} \;\Box$$

Wenn wir eine Funktion $f : \{1, \ldots, n\} \to \mathbb{N}$ als Eingabe eines Problems haben, so ist die *Länge* der Eingabe von der Größenordnung $O(n + \sum_{i=1}^{n} \log(f(i)))$. Wenn wir also von einer polynomiellen Laufzeit sprechen, so meinen wir polynomiell in diesem Wert.

2.2 MAX-DP

Wir betrachten nun eine Klasse von Optimierungsproblemen, die alle in **FPTAS** liegen. Dazu folgende vorbereitende Definition:

Definition 2.4 (schwach monoton)
Sei $k > 0$ und $H : (\mathcal{F}(\mathbb{N}, \mathbb{N}))^k \to \mathbb{N}$ ein Funktional. Dann nennen wir H schwach monoton, *falls für alle $\overline{f} \in \mathcal{F}(\mathbb{N}, \mathbb{N})^{k-1}$ und $f, g, h \in \mathcal{F}(\mathbb{N}, \mathbb{N})$ mit endlichem Träger gilt:*

1. $f \sqsubseteq g \implies H\left(f, \overline{f}\right) \leq H\left(g, \overline{f}\right)$

2. $f \restriction_{Tr(h)} = g \restriction_{Tr(h)} \;\wedge\; H\left(f, \overline{f}\right) \leq H\left(g, \overline{f}\right) \implies H\left(f + h, \overline{f}\right) \leq H\left(g + h, \overline{f}\right) \;\Box$

Bei der Definition wird also eine gewisse Monotonie in der ersten Komponente von H gefordert. Wir benutzen im folgenden die Schreibweise $H(\overline{f})$ auch, wenn die Funktionen $\overline{f}$ nicht auf ganz $\mathbb{N}$, sondern nur auf einer Menge $\{1, \ldots, n\}$ definiert sind. In diesem Falle sei $H(\overline{f}) := H(\overline{f'})$, wobei $\overline{f'}$ die Fortsetzungen der Funktionen $\overline{f}$ auf ganz $\mathbb{N}$ sind, die für Argumente größer als n den Funktionswert 0 annehmen.

Beispiel 2.5 (Schwach monotone Funktionale)
Beispiele schwach monotoner Funktionale H sind:

- $H(f, d) = \sum_i f(i) \cdot d(i)$ (lineares H).

- $H(f) = \sum_i g_i(f(i))$ oder $H(f) = \prod_i g_i(f(i))$, wobei die g_i beliebige nicht-negative, monoton wachsende Funktionen sind.

- H h-monoton (siehe Definition 2.9).

- H Summe oder Produkt schwach monotoner Funktionale. $\Box$

Nun zur Definition der Klasse, die uns in diesem Kapitel beschäftigen wird.

Definition 2.6 (MAX-DP)
MAX-DP *enthält für jedes schwach monotone, in polynomieller Zeit berechenbare*[1] *k-stellige Funktional H das folgende Optimierungsproblem* DP_H.

$\mathrm{DP}_H = (I, F, cost, \max)$

I ist die Menge der $(k+2)$-Tupel $(b, m, c, f_1, \ldots, f_{k-1})$, wobei $b \in \mathbb{N}$ und $m, c, f_1, \ldots, f_{k-1}$ Funktionen von $\{1, \ldots, n\}$ nach $\mathbb{N}$ sind, wobei $\mathbf{1} \sqsubseteq m$, d.h. $\forall i\ m(i) \geq 1$, gilt.

$F((b, m, c, \overline{f}))$ ist die Menge der Funktionen $s : \{1, \ldots, n\} \to \mathbb{N}$ mit $s \sqsubseteq m$ und $H(s, \overline{f}) \leq b$.

$cost((b, m, c, \overline{f}), s) = \sum_{i=1}^{n} s(i) \cdot c(i)$

$\square$

Das Ziel dieses Kapitel ist der Beweis des folgenden Satzes:

Satz 2.7
MAX-DP $\subseteq$ FPTAS $\square$

Daraus ergibt sich sofort die Zugehörigkeit zu **FPTAS** für das Problem Knapsack. Und zwar mittels dem linearen Funktional $H_{\text{Knapsack}}(s, g) := \sum_i s(i) \cdot g(i)$ (wobei g die Gewichtsfunktion ist), $c := w$ (die Wertfunktion) und $m := \mathbf{1}$ (konstant der Wert 1).

Multiple Choice Knapsack, die Variante von Knapsack, bei der Gegenstände mehrfach vorkommen können, erhält man, indem man die Funktion m so bestimmt, daß $m(i)$ angibt, wie oft Gegenstand i vorhanden ist.

Aber auch für ungewöhnliche Knapsack-Varianten mit Kostenfunktionen $\sum_i s(i)^2 \cdot g(i)$ oder $\sum_i g(i)^{s(i)}$ erhält man sofort die Zugehörigkeit zu **FPTAS**.

2.3 Der Beweis

2.3.1 Vorbemerkungen

Bevor wir zum Beweis von Satz 2.7 kommen, machen wir uns klar, daß wir oBdA davon ausgehen können, daß alle Eingaben $(b, m, c, f_1, \ldots, f_k)$ für ein Problem DP_H die folgenden Bedingungen (2.1) und (2.2) erfüllen.

[1] Zur Erinnerung, damit meinen wir: Für alle $f_1, \ldots, f_k$ mit $Tr(f_i) \subseteq \{1, \ldots, n\}$ ist $H(f_1, \ldots, f_k)$ in in n und $\sum_{i=1}^{k} \sum_{j=1}^{n} \log(f_i(j))$ polynomieller Zeit zu berechnen.

$$\forall\ i \quad c(i) \neq 0 \tag{2.1}$$

Denn gilt etwa, im Gegensatz zu (2.1), für ein i_0 die Gleichung $c(i_0) = 0$, so geht der Wert $s(i_0)$ nicht in die Kostenfunktion ein. Haben wir also eine Lösung s, so hat die Funktion s', die mit s bis auf die Stelle i_0 übereinstimmt, und dort den Wert 0 annimmt, die gleichen Kosten. Zudem ist sie eine Lösung des Problems, da wegen der Monotonie von H $H(s', \overline{f}) \leq H(s, \overline{f})$ gilt. Durch Festlegen von $s(i_0) = 0$ können wir bei der folgenden Optimierung mit einem kleineren Träger $\{1, \ldots, n\} - \{i_0\}$ arbeiten.

$$\forall\ i \quad H\left(m(i) \cdot \mathbf{1}_i, \overline{f}\right) \leq b \tag{2.2}$$

Hierbei ist $\mathbf{1}_i$ die Funktion, die an der Stelle i den Wert 1 annimmt, und auf allen anderen Zahlen den Wert 0. Wird Ungleichung (2.2) für ein i_0 nicht erfüllt, so kann wegen der Monotonie von H keine Lösungsfunktion s an der Stelle i_0 den Wert $m(i)$ annehmen. Genauer gesagt, ändern sich die möglichen Lösungen des Problems nicht, wenn man von m zu m' übergeht, mit:

$$m'(i) := \begin{cases} m(i), & \text{falls } i \neq i_0, \\ \text{größte Zahl } p \text{ mit } H(p \cdot \mathbf{1}_{i_0}, \overline{f}) \leq b, & \text{falls } i = i_0 \end{cases}$$

Die entsprechende Zahl p läßt sich mittels binärer Suche im Intervall $\{1, \ldots, m(i)\}$ in polynomieller Zeit bestimmen.[2]

Nun aber zum Beweis unseres Satzes. Er basiert auf einer Anwendung des Algorithmenentwurfsprinzips "dynamische Programmierung" (deshalb auch der Name **MAX-DP**).[3]

Beweis (Satz 2.7):

Sei H ein schwach monotones, k-stelliges, in polynomieller Zeit berechenbares Funktional. Wir konstruieren nun einen Algorithmus, der für eine Fehlerschranke $\varepsilon > 0$ und eine Eingabe $\mathcal{I} = (b, m, c, f_1, \ldots, f_k)$ der Größe $N := |\mathcal{I}|$ in Zeit $O((\frac{N}{\varepsilon})^{O(1)})$ eine ε-approximative Lösungsfunktion s ermittelt. Die Größe des Trägers der Eingabefunktionen bezeichnen wir mit n.

---

[2] D.h. man testet zunächst $H(\frac{1}{2}m(i) \cdot \mathbf{1}_{i_0}, \overline{f}) \leq b$. Wenn das gilt, fährt man mit $\frac{3}{4}m(i)$, sonst mit $\frac{1}{4}m(i)$ fort, usw. Durch dieses wiederholte Halbieren des untersuchten Intervalls ist man nach spätestens $\lceil \log m(i) \rceil$ Tests am Ziel.

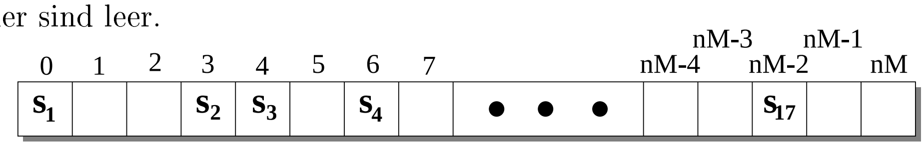
[3] Im Prinzip ist er eine Verallgemeinerung des klassischen Beweises für Knapsack $\in$ **FPTAS**.

Diese Konstruktion vollzieht sich in zwei Schritten: zuerst zeigen wir, wie man in exponentieller Zeit die optimale Lösung des Problems finden kann. Anschließend modifizieren wir diesen Algorithmus so, daß er in polynomieller Zeit die gewünschte ε-approximative Lösung findet.

Im folgenden sei $M := \max_{1 \leq i \leq n} c(i) \cdot m(i)$. Da alle Lösungen s die Ungleichung $s \sqsubseteq m$ erfüllen, gilt für sie $cost(s) \leq n \cdot M$ (hier wie im folgenden lassen wir das erste Argument der Kostenfunktion (die Eingabe) weg – der Übersichtlichkeit wegen).

2.3.2 Optimale Lösung

Der Algorithmus, der die optimale Lösung berechnet, bedient sich eines Arrays A, das von 0 bis $n \cdot M$ indiziert ist. Während der Ausführung des Algorithmus enthalten die Arrayelemente $A(i)$ entweder eine Lösungsfunktion s, oder sind leer.



Die Funktionsweise des Algorithmus läßt sich halb-formal wie folgt beschreiben:

1. Zu Beginn ist das gesamte Array leer, nur an der Stelle 0 steht die Funktion $\mathbf{0}$ (konstant $= 0$).

2. `FOR` i `:= 1 TO` n

   (a) `FOR` p `:=` nM `DOWNTO 0`

   (b) Enthält $A(p)$ eine Funktion s_0, dann
   `FOR` j `:= 1 TO` $m(i)$

      i. s_j `:=` s_0 `+` $j \cdot \mathbf{1}_i$

      ii. `cst :=` $cost(s_j)$

      iii. Falls $A(cst)$ leer oder $H(A(cst)) < H(s_j)$, dann
      `A(cst) :=` s_j

3. Wir geben von denen nun im Array stehenden Funktionen s mit $H(s) \leq b$ diejenige aus, die an der höchsten Stelle steht.

Zunächst ist klar, daß der Algorithmus immer terminiert. Die Laufzeit können wir wie folgt abschätzen. Der Algorithmus besteht aus drei Schleifen, die n, nM, bzw. $m(i)$-mal durchlaufen werden. Im Inneren der Schleifen wird im wesentlichen einmal H ausgewertet. Dies liefert insgesamt eine Laufzeit von $O(n^2M^2N^{O(1)})$. Da M exponentiell in N groß sein kann, ist die Laufzeit im allgemeinen exponentiell. Weiterhin gilt:

Lemma 2.8
Der Algorithmus ist korrekt, d.h. liefert immer eine optimale Lösung s.

Beweis: Wir zeigen induktiv, daß nach dem i-ten Durchlauf der äußeren Schleife folgende Invariante (*) erfüllt ist.

> Für jede Zahl $p \in \{0, 1, \ldots, nM\}$ gilt: $A(p)$ enthält unter allen Funktionen s mit $Tr(s) \subseteq \{1, \ldots, i\}$ und $cost(s) = p$ diejenige mit geringsten Wert $H(p)$. (*)

Aus der Gültigkeit der Invariante für $i = n$ folgt dann die Behauptung.

Für $i = 0$ gilt (*) trivialerweise , da es nur eine Funktion mit $Tr(f) \subseteq \emptyset$ gibt: die Nullfunktion $\mathbf{0}$. Für diese gilt auch $cost(\mathbf{0}) = 0$, wie gefordert.

Gelte (*) für ein i mit $0 \leq i < n$. Wir zeigen, daß sie dann auch für $i + 1$ gilt. Sei dazu $p \in \{0, \ldots, nM\}$ beliebig, und s eine Funktion, die den geringsten Wert $H(s)$ unter allen Funktionen mit $cost(s) = p$ und $Tr(f) \subseteq \{1, \ldots, i+1\}$ hat. Wir wollen zeigen, daß nach dem $(i+1)$-ten Durchlauf der äußeren Schleife in $A(p)$ eine Funktion s' mit $H(s) = H(s')$ steht.

Sei $t := s - s(i+1) \cdot \mathbf{1}_{i+1}$, d.h. die Funktion die mit s bis auf die Stelle $i + 1$ übereinstimmt, und dort gleich 0 ist. Nach Induktionsvoraussetzung steht nach dem i-ten Durchlauf der Schleife an der Stelle $cost(t)$ des Arrays eine Funktion t', für die $H(t') \leq H(t)$ gilt.

Für $s' := t' + s(i+1) \cdot \mathbf{1}_{i+1}$ gilt offenbar $cost(s') = cost(s) = p$, und nach Bedingung 2 der Definition von 'schwach monoton'

$$\begin{array}{rl} & H(t') \leq H(t) \;\wedge\; i+1 \not\in Tr(t), Tr(t') \\ \Longrightarrow & H(s') \;=\; H(t' + \mathbf{1}_{i+1}) \;\leq\; H(t + \mathbf{1}_{i+1}) \;=\; H(s) \end{array}$$

Wegen der angenommenen Minimalität von $H(s)$, gilt somit $H(s) = H(s')$.

Da t' nach den i-ten Durchlauf im Array steht, wird es beim $(i+1)$-ten Durchlauf betrachtet. Insbesondere wird s' an der Position p eingetragen, es sei denn, dort steht schon eine andere optimale Funktion. Auf jeden Fall wird also die Invariante (*) an der (beliebig gewählten) Stelle p nach dem Schleifendurchlauf erfüllt sein. ■

2.3.3 Approximative Lösung

Warum hat der Algorithmus exponentielle Laufzeit? Weil wir ein Array mit Einträgen für alle möglichen Werte der Kostenfunktion $\sum_{i=1}^{n} s(i) \cdot c(i)$ verwenden, und die Kostenfunktion exponentiell viele mögliche Werte hat. Der

'Trick' im folgenden ist es, das Problem leicht zu modifizieren, so daß die Kostenfunktion nur noch polynomiell viele Werte annimmt. Dazu verwenden wir die Standardtechnik des 'Weglassen der niedrigstwertigsten Bits'.

Hierzu setzen wir (man beachte Ungleichung (2.1))

$$a(i) = \left\lceil \log\left(\frac{M}{m(i)} \cdot \frac{\varepsilon}{4n}\right)\right\rceil, \; b(i) = \left\lceil \log\left(\frac{M}{c(i)} \cdot \frac{\varepsilon}{4n}\right)\right\rceil$$

und ändern den Algorithmus wie folgt leicht ab:

1. Zu Beginn des Algorithmus runden wir alle Werte $c(i)$ auf das nächstkleinere Vielfache von $2^{a(i)}$ ab.

2. Bei den Lösungsfunktionen s lassen wir nur Funktionswerte $s(i)$ zu, die Vielfache von $2^{b(i)}$ sind.

Unter diesen Einschränkungen sind die in der Kostenfunktion $\sum_{i=1}^{n} s(i) \cdot c(i)$ vorkommenden Summanden $s(i) \cdot c(i)$ Vielfache von

$$2^{b(i)} \cdot 2^{a(i)} \geq \frac{M\varepsilon}{4n \cdot c(i)} \cdot \frac{M\varepsilon}{4n \cdot m(i)} = \frac{M^2\varepsilon^2}{16n^2 m(i)c(i)} \geq \frac{M^2\varepsilon^2}{16n^2 M} = \frac{M\varepsilon^2}{16n^2}$$

Alle möglichen Werte der Kostenfunktion $\sum_{i=1}^{n} s(i) \cdot c(i)$ sind somit Vielfache einer Zweierpotenz $\geq \frac{M\varepsilon^2}{16n^2}$. Die Anzahl ihrer möglichen Werte ist daher höchstens gleich

$$\frac{nM}{\frac{M\varepsilon^2}{16n^2}} = \frac{16n^3}{\varepsilon^2} = O\left(\left(\frac{n}{\varepsilon}\right)^{O(1)}\right)$$

Die Größe des Arrays ist damit polynomial in n und $\frac{1}{\varepsilon}$. Die Gesamtlaufzeit des Algorithmus ist dann $O(\frac{N}{\varepsilon})^{O(1)})$, wobei das N durch das Einlesen und Runden der Daten sowie zum Berechnen des Funktionals H hinzukommt. Insgesamt haben wir also ein FPTAS.

Es fehlt noch der Nachweis, daß die von dem Algorithmus berechnete Lösung ε-approximativ ist. Sei s_{opt} eine optimale Lösung des ursprünglichen Problems. Weiterhin sei s_{red} die Funktion, die man aus s_{opt} erhält, wenn man alle Funktionswerte auf das nächstkleinere Vielfache von $2^{b(i)}$ abrundet:

$$s_{red}(i) := 2^{b(i)} \cdot \left\lfloor \frac{s_{opt}(i)}{2^{b(i)}} \right\rfloor$$

Da s_{red} eine mögliche Lösung unseres modifizierten Problems ist, wird der Algorithmus eine Lösung finden, die mindestens so gut ist, d.h. wenigstens die folgenden Kosten hat:

$$\sum_{i=1}^{n} s_{red}(i) \cdot \left(2^{a(i)} \left\lfloor \frac{c(i)}{2^{a(i)}} \right\rfloor\right) \geq \sum_{i=1}^{n} \left(s_{opt}(i) - 2^{b(i)}\right) \cdot \left(c(i) - 2^{a(i)}\right)$$

Das bedeutet aber, daß s_{red} schon eine ε-approximative Lösung ist, wie man durch Abschätzen des relativen Fehlers sieht.

$$\text{rel. Fehler} \leq \frac{\sum_{i=1}^n s_{opt}(i)c(i) - \sum_{i=1}^n \left(s_{opt}(i) - 2^{b(i)}\right)\left(c(i) - 2^{a(i)}\right)}{\sum_{i=1}^n s_{opt}(i)c(i)}$$
$$\leq \frac{\sum_{i=1}^n \left(2^{b(i)}c(i) + 2^{a(i)}m(i) - 2^{a(i)+b(i)}\right)}{\sum_{i=1}^n s_{opt}(i)c(i)} \overset{(+),s.u.}{\leq} \sum_{i=1}^n \frac{\left(2^{b(i)}c(i) + 2^{a(i)}m(i)\right)}{M}$$
$$= \sum_{i=1}^n \left(2^{b(i)}\frac{c(i)}{M} + 2^{a(i)}\frac{m(i)}{M}\right)$$
$$\leq \sum_{i=1}^n \left(2 \cdot \frac{M}{c(i)} \cdot \frac{\varepsilon}{4n} \cdot \frac{c(i)}{M} + 2 \cdot \frac{M}{m(i)} \cdot \frac{\varepsilon}{4n} \cdot \frac{m(i)}{M}\right) = \sum_{i=1}^n \frac{\varepsilon}{n} = \varepsilon$$

Die Umformung (+) benutzt dabei die Tatsache $\sum_{i=1}^n s_{opt}(i)c(i) \geq M = \max_{1\leq i\leq n} m(i)c(i)$. Dies stimmt, da wegen Ungleichung (2.2) jedes $m(i) \cdot \mathbf{1}_i$ eine mögliche Lösung des Problems ist. ■

2.4 Malmströms Resultat

Als Sonderfall folgt aus dem Resultat **MAX-DP** $\subseteq$ **FPTAS** ein Satz, den Malmström in [Malmström96a] und [Malmström96b] gezeigt hat. Sein Resultat wurde unter Verwendung von Begriffen der "meta-endlichen Modelltheorie" (metafinite model theory) formuliert und bewiesen. Wir geben eine äquivalente Formulierung seines Satzes, die ohne diesen Apparat auskommt. Zunächst jedoch eine Hilfsdefinition.

Definition 2.9 (h-monoton)
Ein Funktional $H : \mathcal{F}(\mathbb{N}^q, \mathbb{N})^k \to \mathbb{N}$ *heißt* h-monoton, *falls für alle* $\overline{f} \in \mathcal{F}(\mathbb{N}^q, \mathbb{N})^{k-1}$ *und* $f, g, h \in \mathcal{F}(\mathbb{N}^q, \mathbb{N})$ *gilt:*

1. $f \sqsubset g \implies H\left(f, \overline{f}\right) < H\left(g, \overline{f}\right)$
2. $H\left(f, \overline{f}\right) \leq H\left(g, \overline{f}\right) \implies H\left(f + h, \overline{f}\right) \leq H\left(g + h, \overline{f}\right)$ □

Analog zu schwach monotonen Funktionen benutzen wir die Schreibweise $H(f)$ auch für Funktionen f mit endlichem Definitionsbereich. Man beachte, daß im Falle $q = 1$ jedes h-monotone Funktional auch schwach monoton ist. Definition 2.9 ist also restriktiver als Definition 2.4.

Bei Malmströms Optimierungsproblemen sind die Eingaben meta-endliche Strukturen, d.h. im wesentlichen endliche, relationale Strukturen, auf denen

$\mathbb{N}$-wertige Funktionen erklärt sein können.[4] Die folgende Definition der Problemklasse bildet dies ohne die explizite Verwendung des Begriffes 'meta-endliche Struktur' nach.

Definition 2.10 (MAX Φ)
MAX Φ *enthält für*

- *jedes h-monotone, in polynomieller Zeit berechenbare Funktional H und*
- *jede Formel $\psi = \forall \overline{x} \varphi(\overline{x}, s(\overline{x}))$, wobei $\overline{x}$ ein q-Tupel und $\varphi(\overline{x}, y)$ eine in polynomieller Zeit auswertbare Formel ist, die monoton in y ist (d.h. es gilt $\varphi(\overline{x}, a) \implies \forall y \leq a\ \varphi(\overline{x}, y)$).*

das folgende Optimierungsproblem $\mathrm{DP}_{H,\psi}$.

$\mathrm{DP}_{H,\psi} = (I, F, cost, \max)$

I ist die Menge der $(k+2)$-Tupel $(\mathfrak{A}, b, c, f_1, \ldots, f_{k-1})$, wobei $\mathfrak{A} = (A, \overline{R})$ (in zu ψ passender Signatur, und $A = \{1, \ldots, n\}$), $b \in \mathbb{N}$, und $c, f_1, \ldots, f_{k-1}$ Funktionen von A^q nach $\mathbb{N}$ sind.

$F((\mathfrak{A}, b, c, \overline{f}))$ ist die Menge der Funktionen $s : A^q \to \mathbb{N}$ mit $(\mathfrak{A}, s) \models \psi$ und $H(s, \overline{f}) \leq b$.

$cost((\mathfrak{A}, b, c, \overline{f}), s) =$

$$\sum_{\overline{a} \in A^q} s(\overline{a}) \cdot c(\overline{a})$$

□

Das 'Φ' in **MAX Φ** hat meines Wissens übrigens keine besondere Bedeutung.

Satz 2.11 (Malmström 1996)
MAX Φ $\subseteq$ **FPTAS**.

Beweis:
Das folgt direkt aus Satz 2.7, denn wir können jedes Problem aus **MAX Φ** in polynomieller Zeit zu einem lösungsäquivalenten Problem in **MAX-DP** umformen. Sei ein konkretes Problem $\mathrm{DP}_{H,\psi}$ aus **MAX Φ** gegeben. OBdA sei $q = 1$, H also schwach monoton.[5] Sei weiterhin eine Eingabe $(\mathfrak{A}, b, c, f_1, \ldots, f_{k-1})$ gegeben.

[4] Die genaue Definition ist etwas komplizierter, würde an der restlichen Diskussion aber nicht viel ändern; deshalb verzichten wir darauf.

[5] Bei $q > 1$ muß man $A^q = \{1, \ldots, n\}^q$ zunächst mit $\{1, \ldots, n^q\}$ identifizieren, da schwach monotone Funktionale im Gegensatz zu h-monotonen nicht für Funktionen auf Produkträumen definiert sind. Der Rest verläuft analog.

Wir zeigen nun, wie man in polynomieller Zeit eine Funktion m konstruieren kann, so daß $(b, m, c, f_1, \ldots, f_{k-1})$ eine Eingabe für DP_H mit den gleichen Lösungen ist. Da die Kostenfunktion ebenfalls dieselbe ist, folgt dann aus $\mathrm{DP}_H \in \mathbf{FPTAS}$ (Satz 2.7) sofort die Behauptung $\mathrm{DP}_{H,\psi} \in \mathbf{FPTAS}$.

Um m zu bestimmen, bemerken wir zunächst, daß aus der ersten Bedingung der Definition h-monotoner Funktionen für alle $i \in \{1, \ldots, n\}$ folgt

$$0 \leq H(0) < H(\mathbf{1}_i) < H(2 \cdot \mathbf{1}_i) < \cdots < H((b+1) \cdot \mathbf{1}_i) \implies b < H((b+1) \cdot \mathbf{1}_i)$$

Für alle Lösungsfunktionen s muß daher $\forall i \; s(i) \leq b$ gelten. Folgende Funktion m leistet also das gewünschte:

$$m(i) = \begin{cases} b, & \text{falls } \mathfrak{A} \models \varphi(i, b), \\ \text{das größte } p \text{ mit } \mathfrak{A} \models \varphi(i, p), & \text{falls } \mathfrak{A} \not\models \varphi(i, b) \end{cases}$$

Hierbei läßt sich im zweiten Fall das p mittels binärer Suche im Intervall $\{1, \ldots, b\}$ in polynomieller Zeit bestimmen.

Damit ist $\mathrm{DP}_{H,\psi}$ auf DP_H reduziert, was den Satz beweist. ■

Kapitel 3

PTAS

Auch für die Komplexitätsklasse **PTAS** stellen wir uns die Frage, ob sie eine syntaktische Charakterisierung hat. Ein erster Schritt findet sich in der Dissertation von Sanjeev Khanna [Khanna96]. Er definierte mehrere Probleme, die Varianten von MAX SAT sind, und in **PTAS** liegen. Viele bekannte Probleme aus **PTAS** lassen sich auf diese Probleme reduzieren.

Wir werden nun zunächst Khanna's Probleme definieren und dann PTAS-Algorithmen für sie konstruieren, was komplizierter sein wird als für **MAX-DP** im letzten Kapitel.

3.1 Probleme

Die drei Probleme, die uns für den Rest des Kapitels beschäftigen, heißen TMAX, TMIN und MPSAT. Die Eingaben der Probleme sind Mengen boolescher Formeln in disjunktiver Normalform. Bevor wir die Probleme selbst definieren, führen wir noch ein paar Begriffe ein.

Definition 3.1 (Implikant)
Sei φ eine boolesche Formel in disjunktiver Normalform, d.h. $\varphi = \bigvee_{i=1}^{k} \psi_i$, mit $\psi_i = x_{i_1} \wedge \cdots \wedge x_{i_l}$. Dann nennen wir die ψ_i Implikanten *von φ (da ihre Gültigkeit die Gültigkeit von φ impliziert).* $\square$

Definition 3.2 (Interpretation, $\models$, $\not\models$)
Sei φ eine boolesche Formel in den Variablen $x_1, \ldots, x_m$. Wir nennen Funktionen $\mathfrak{I} : \{x_1, \ldots, x_m\} \to \{0, 1\}$ Interpretationen *oder Belegungen der Variablen $x_1, \ldots, x_m$.*

Wir schreiben $\mathfrak{I} \models \varphi$ (bzw. $\mathfrak{I} \not\models \varphi$) falls bei einer Belegung der Variablen gemäß der Interpretation $\mathfrak{I}$ (wobei wir 0 mit 'falsch', und 1 mit 'wahr' identifizieren), die Formel φ erfüllt (bzw. nicht erfüllt) wird. $\square$

Nun können wir unsere drei Probleme definieren.

Definition 3.3 (TMAX, TMIN)
TMAX *und* TMIN *sind die folgenden Optimierungprobleme:*

TMAX$= (B_{neg}, I, cost, \max)$, *und*
TMIN$= (B_{pos}, I, cost, \min)$, *wobei*

B_{neg} *(bzw. B_{pos}) besteht aus den Tripeln $(\{\varphi_1, \ldots, \varphi_n\}, \{x_1, \ldots, x_m\}, w)$, wobei die φ_i boolesche Formeln in disjunktiver Normalform in den Variablen $\{x_1, \ldots, x_m\}$ sind, in denen alle Variablen* **neg***ativ (bzw.* **pos***itiv) vorkommen. w ist eine Funktion von $\{x_1, \ldots, x_m\}$ nach $\mathbb{N}$.*

$I((\overline{\varphi}, \overline{x}, w))$ *besteht aus denjenigen Interpretationen $\mathfrak{I}$ der Variablen $x_1, \ldots, x_m$, die alle Formeln φ_i erfüllen ($\forall i\ \mathfrak{I} \models \varphi_i$).*

$cost((\overline{\varphi}, \overline{x}, w), \mathfrak{I}) = \sum_{i, \mathfrak{I}(x_i)=1} w(x_i)$

□

Man beachte, daß im Falle von TMAX die Interpretation $\mathfrak{I} \equiv 0$ (konstant gleich 0) bzw. für TMIN die Interpretation $\mathfrak{I} \equiv 1$ immer alle Formeln erfüllt, da die Variablen nur negativ bzw. positiv in den φ_i vorkommen. Also ist die Lösungsmenge für keine Eingabe $(\overline{\varphi}, \overline{x}, w)$ leer, die Probleme also wohldefiniert.

Definition 3.4 (MPSAT)
MPSAT *ist das folgende Optimierungsproblem:*

MPSAT$= (B, I, cost, \max)$, *wobei*

B *besteht aus den Tupeln $(\{\varphi_1, \ldots, \varphi_n\}, \{x_1, \ldots, x_m\}, w, b)$, wobei die φ_i boolesche Formeln in disjunktiver Normalform in den Variablen $\{x_1, \ldots, x_m\}$ sind. w ist eine Funktion von $\{\varphi_1, \ldots, \varphi_n, x_1, \ldots, x_m\}$ nach $\mathbb{N}$, und $b \in \mathbb{N}$.*

$I((\overline{\varphi}, \overline{x}, w, b))$ *besteht aus denjenigen Interpretationen $\mathfrak{I}$ der Variablen $x_1, \ldots, x_m$, die folgende Ungleichung erfüllen:*

$$\sum_{i, \mathfrak{I}(x_i)=1} w(x_i) \leq b$$

$cost((\overline{\varphi}, \overline{x}, w, b), \mathfrak{I}) =$
$$\sum_{i, \mathfrak{I} \models \varphi_i} w(\varphi_i)$$

□

Auch hier ist die Interpretation $\mathfrak{I} \equiv 0$ immer eine Lösung.

Die drei Probleme, die wir soeben definiert haben, liegen noch nicht in **PTAS**.[1] Das erreichen wir, indem wir die möglichen Eingaben noch etwas einschränken.[2]

Definition 3.5 (Abhängigkeitsgraph $G_\mathcal{I}$)
Sei $\mathcal{I} = (\{\varphi_1, \ldots, \varphi_n\}, \{x_1, \ldots, x_m\}, w)$ *bzw.* $\mathcal{I} = (\{\varphi_1, \ldots, \varphi_n\}, \{x_1, \ldots, x_m\}, w, b)$ *eine Eingabe zu* TMAX, TMIN *bzw.* MPSAT.

Der Abhängigkeitsgraph *für* $\mathcal{I}$ *ist der Graph* $G_\mathcal{I} = (V, E)$ *mit*

- $V = \{\varphi_1, \ldots, \varphi_n, x_1, \ldots, x_m\}$
- $E = \{(x_i, \varphi_j), (\varphi_j, x_i) \mid x_i \textit{ kommt in } \varphi_j \textit{ vor } \}$ $\square$

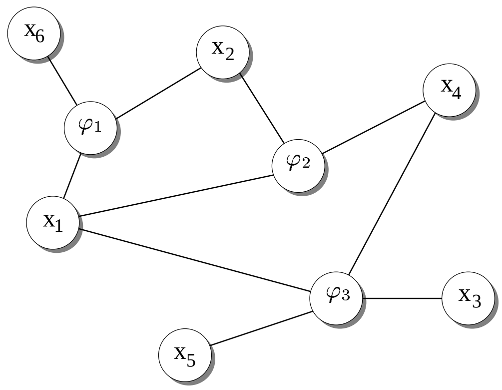
Ein Beispiel: für die Formeln

$$\varphi_1 = x_1 \lor (\neg x_2 \land x_6) \lor (x_1 \land \neg x_2)$$

$$\varphi_2 = (x_2 \land \neg x_4) \lor (x_1 \land x_2)$$

$$\varphi_3 = (\neg x_1 \land x_3 \land x_4 \land \neg x_5) \lor x_4$$

sieht der Abhängigkeitsgraph so aus:



Der Abhängigkeitsgraph ist hier offenbar planar. Wenn wir die Eingaben unserer drei Probleme so einschränken, daß die Abhängigkeitsgraphen planar sind, dann liegen die Probleme in **PTAS**.

[1] Immer unter der Annahme $\mathbf{P} \neq \mathbf{NP}$. Denn MAX-SAT ist ein Sonderfall von MPSAT.

[2] In [KSW97] und [KST97] liefern Khanna et al. eine interessante Klassifikation von Problemen ähnlichen Typs wie unsere drei Probleme, abhängig von einschränkenden Bedingungen, die die Formeln φ_i erfüllen. Die Art der betrachteten Einschränkungen unterscheidet sich jedoch deutlich von unserer, so daß die beiden Resultate unabhängig voneinander sind.

Definition 3.6 (PLANAR TMAX, PLANAR TMIN, PLANAR MPSAT)
Die PLANAR TMAX, PLANAR TMIN *und* PLANAR MPSAT *sind die Probleme, die man aus* TMAX, TMIN *und* MPSAT *erhält, wenn man nur Eingaben $\mathcal{I}$ zuläßt, deren Abhängigkeitsgraph $G_{\mathcal{I}}$ planar ist.*[3] □

Satz 3.7 (Khanna 1996 [Khanna96, KhMo96])
PLANAR TMAX, PLANAR TMIN, PLANAR MPSAT $\in$ **PTAS** □

3.2 Beispiele

Bevor wir zu dem Beweis von Satz 3.7 kommen, zunächst einige Beispiele, die zeigen, daß sich recht natürliche Probleme auf PLANAR TMAX, PLANAR TMIN und PLANAR MPSAT reduzieren lassen.

3.2.1 KNAPSACK

KNAPSACK läßt sich auf PLANAR MPSAT reduzieren.

Für jedes der n Objekte haben wir je eine Variable x_i und eine Formel $\varphi_i = x_i$. Der zugehörige Graph ist offensichtlich planar: er besteht aus n 'einzelnen' Kanten.

Wir setzen $w(x_i) := g(i)$ (das Gewicht des i-ten Gegenstandes) und $w(\varphi_i) := w(i)$ (der Wert des i-ten Gegenstandes). b ist das größte zulässige Gesamtgewicht.

3.2.2 PLANAR MAXIMUM INDEPENDENT SET

PLANAR MAXIMUM INDEPENDENT SET läßt sich auf PLANAR TMAX reduzieren.

Denn sei ein Graph $G = (V, E)$ gegeben. Die entsprechende Eingabe für PLANAR TMAX sieht wie folgt aus. Für jeden Knoten $v \in V$ des Graphen haben wir eine Variable x_v, und für jede Kante $(v, w) \in E$ eine Formel $\varphi_{(v,w)} = \neg x_v \vee \neg x_w$. Der zugehörige Abhängigkeitsgraph ist planar, solange G planar ist (siehe Abbildung 3.1: jeder Kante wird ein Knoten hinzugefügt).

Die Belegung der Variablen x_v gibt an, ob der Knoten v zum 'independent set' gehört, und die Formeln stellen sicher, daß nicht zwei benachbarte Knoten beide zu dieser Menge gehören.

[3] Man beachte, daß sich in linearer Zeit prüfen läßt, ob ein Graph planar ist [HoTa74]. Die Probleme bleiben also in **NPO**.

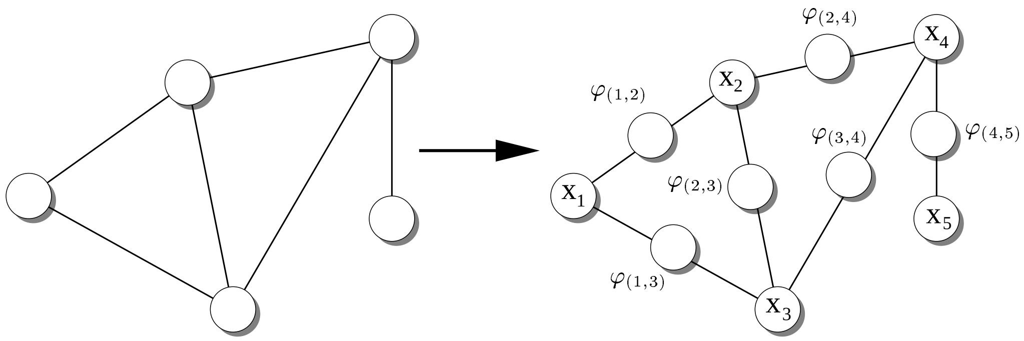


Abbildung 3.1: Planar Maximum Independent Set: Graph (links), zugehöriger Abhängigkeitsgraph (rechts)

3.2.3 Planar Minimum Vertex Cover

Bei Planar Minimum Vertex Cover ist es das Ziel, gegeben einen planaren Graphen $G = (V, E)$, die kleinstmögliche Teilmenge $T \subseteq V$ zu finden, so daß für jede Kante $(v, w) \in E$ zumindest einer der Endpunkte in T ist.

Dieses Problem läßt sich leicht in Planar TMIN einbetten. Wir haben für jeden Knoten $v \in V$ eine Variable x_v und für jede Kante $(v, w) \in E$ eine Formel $\varphi_{(v,w)} = x_v \vee x_w$. Dann entsprechen sich $v \in T$ und die Belegung $\mathfrak{I}(x_v) = 1$.

3.2.4 Planar MAX-SAT

Planar MAX-SAT, das man aus dem regulären MAX-SAT ([GaJo79], Problem LO1) erhält, wenn man fordert, daß der Abhängigkeitsgraph der Eingaben planar ist, ist ein Sonderfall von Planar MPSAT (man setze $b = +\infty$).

3.3 p-außerplanare Graphen

Unser Beweis von Satz 3.7 benutzt entscheidend eine Entdeckung von Brenda Baker [Baker83, Baker94]. Zentral dabei ist der Begriff von *p-außerplanaren Graphen.*

Definition 3.8 (p-außerplanar)
Wir nennen einen Graph G 1-außerplanar *(oder einfach* außerplanar*), wenn er eine Einbettung in die Ebene besitzt, in der sich keine zwei Kanten schneiden, und zudem alle Knoten am Rand der Außenfläche liegen.*

Wir nennen einen Graph G p-außerplanar, *wenn er eine Einbettung in die Ebene besitzt, so daß der beim Weglassen aller äußeren Knoten resultierende*

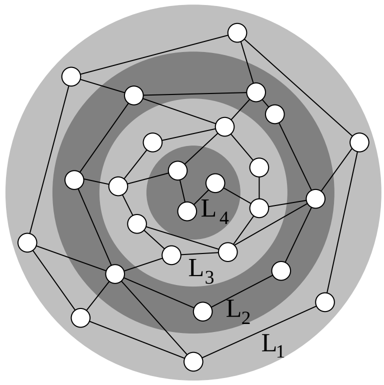


Abbildung 3.2: Ein 4-außerplanarer Graph mit eingezeichneten Schalen L_1, L_2, L_3, L_4.

Graph $(p-1)$*-außerplanar ist.* □

Beispiele 1-außerplanarer Graphen sind Bäume, einfache Zykel oder auch zwei Zykel, die einen Knoten gemeinsam haben.

Anschaulich kann man sich einen p-außerplanaren Graph als einen Graph mit p "Schalen" vorstellen (siehe Abbildung 3.2).

Es zeigt sich, daß viele NP-vollständige Graphen-Probleme auf p-außerplanaren Graphen in polynomieller Zeit lösbar sind.

Baker stellte fest, daß man diese Probleme dann oft auf beliebigen planaren Graphen in **PTAS** liegen. Dazu zerlegt man den Graphen zuerst in mehrere p-außerplanare Graphen, berechnet optimale Lösungen für diese Teilgraphen und setzt die Teillösungen zu einer Gesamtlösung zusammen (Abbildung 3.3).

Baker's Beweis ist leider sehr technisch (siehe [Baker94]). Die elegantere Formulierung von Khanna benutzt eine "Baumzerlegung" des Graphen. Dieser Begriff wurde von Robertson und Seymour im Rahmen ihrer Arbeit über Graphenminoren eingeführt [RoSe83, RoSe86]. Einen guten Einstieg in diese Thematik gibt der Übersichtsartikel von Bodlaender [Bodlaender93b].

3.4 Baumzerlegungen

Definition 3.9 (Baumzerlegung)
Sei $G = (V, E)$ *ein Graph. Eine* Baumzerlegung *von* G *ist ein Paar* $(T = (I, F), \{X_i \mid i \in I\})$*, wobei* $T = (I, F)$ *ein endlicher Baum*[4] *ist und die* X_i

[4] Also ein zusammenhängender, zyklenfreier Graph.

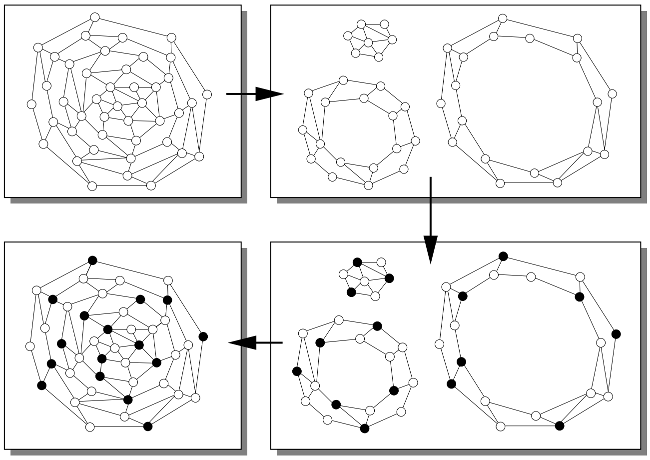


Abbildung 3.3: Im Uhrzeigersinn von links oben: ursprüngliches Problem, Zerlegung in p-außerplanare Teilprobleme, Lösen der Teilprobleme, Zusammensetzen zur Lösung des Ausgangsproblems

Teilmengen von V sind. Zusätzlich gilt:

- $\bigcup_{i \in I} X_i = V$
- *Für jede Kante $(v, w) \in E$ gibt es ein $i \in I$ mit $v, w \in X_i$.*
- *Für jeden Knoten $v \in V$ ist der durch $I_v := \{i \mid v \in X_i\}$ induzierte Teilgraph von T zusammenhängend (und somit ebenfalls ein Baum).* □

Jeder Graph hat eine triviale Baumzerlegung, bei der I aus nur einem Element i besteht, und $X_i = V$ gilt. Ist ein Graph nicht vollständig, so hat er aber auch Baumzerlegungen, bei denen die X_i von V verschieden sind, wie in Abbildung 3.4 (aus [Bodlaender93b]).

Von besonderer Bedeutung sind Graphen, bei denen die X_i nicht zu groß sind.

Definition 3.10 (Breite, Baumbreite)
Sei $G = (V, E)$ ein Graph, und $(T = (I, F), \{X_i \mid i \in I\})$ eine Baumzerlegung von G. Als Breite *der Zerlegung bezeichnen wir den Wert* $\max_{i \in I} |X_i| - 1$.

Die Baumbreite *von G ist die kleinste Breite aller Baumzerlegungen von G.* □

Der Graph in Abbildung 3.4 hat Baumbreite 2. Intuitiv gilt: je mehr Kanten ein Graph enthält, desto größer die Baumbreite. Ein zyklenfreier

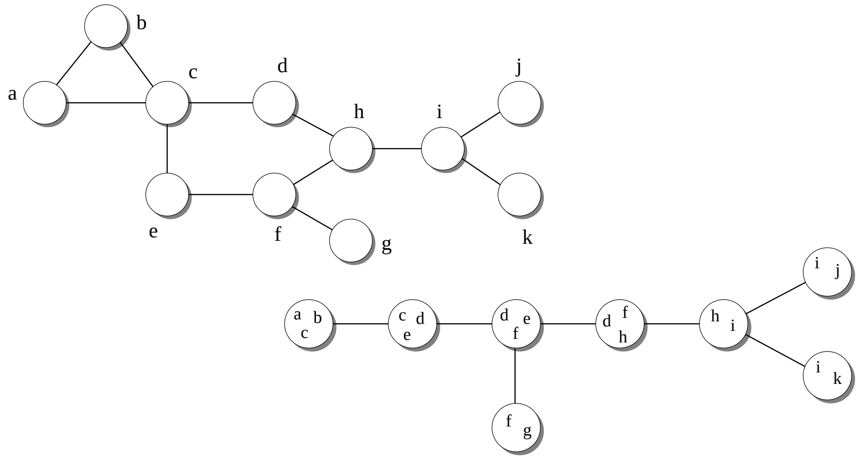


Abbildung 3.4: Graph (oben) mit zugehöriger Baumzerlegung (unten)

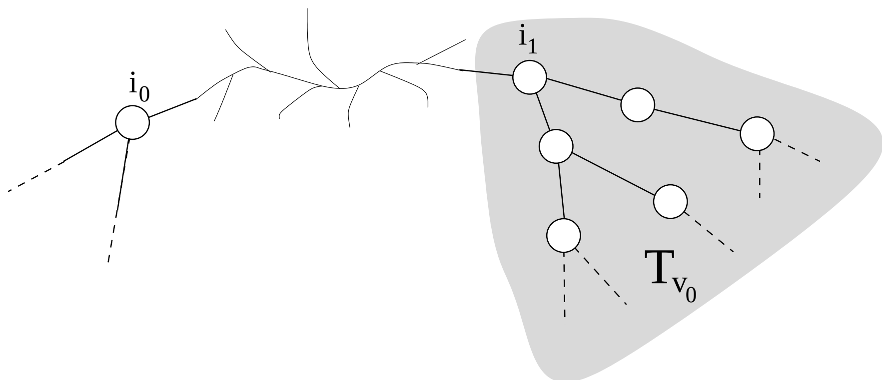


Abbildung 3.5: i_1 ist der Knoten von T_{v_0}, der i_0 am nächsten ist

Graph (ein Baum) hat Baumbreite 1, ein vollständiger Graph der Größe n hat Baumbreite $n-1$. Da dies nicht völlig offensichtlich ist, beweisen wir es kurz.

Bemerkung 3.11
Der vollständige Graph mit n Knoten hat Baumbreite $n-1$.
Beweis: Sei $G=(V,E)$ der vollständige Graph mit n Knoten. Sei weiterhin eine Baumzerlegung $(T=(I,F),\{X_i \mid i \in I\})$ gegeben, und X_{i_0} mit $|X_{i_0}| < n$ sei das größte der X_i. Wegen $|X_{i_0}| < n$ gibt es einen Knoten v_0 mit $v_0 \notin X_{i_0}$.

Sei T_v für jeden Knoten $v \in V$ der durch $I_v := \{i \mid v \in X_i\}$ induzierte Teilbaum von T. Da T ein Baum ist, gibt es einen eindeutig bestimmten Knoten $i_1 \in I_{v_0}$, der i_0 am nächsten liegt (siehe Abbildung 3.5).

Nun liegt i_1 auf dem Pfad von i_0 zu jedem Element von T_{v_0}. Da für alle $v \in X_{i_0}$ aber $I_v \cap I_{v_0} \neq \emptyset$ gilt (wegen $(v, v_0) \in E$), folgt $v \in X_{i_1}$.

Somit haben wir $X_{i_0} \cup \{v_0\} \subseteq X_{i_1}$, und X_{i_1} enthält mindestens ein Element mehr als X_{i_0} $\Longrightarrow$ Widerspruch zur Definition von i_0! ■

Kommen wir nun zu der Verbindung zu p-außerplanaren Graphen.

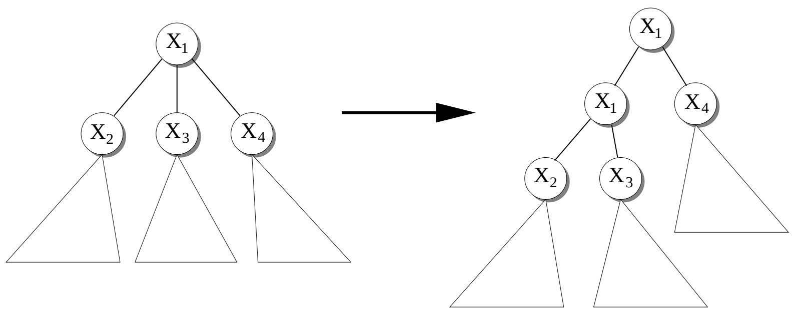
Satz 3.12 (Bodlaender 1988 [Bodlaender88])
Ein p-außerplanarer Graph hat Baumbreite $\leq 3p - 1$. □

Satz 3.13 (Bodlaender 1993 [Bodlaender93a])
Für festes k läßt sich zu allen Graphen mit Baumbreite $\leq k$ in linearer Zeit eine Baumzerlegung mit Breite $\leq k$ finden.[5] □

Lemma 3.14
Wenn wir eine Baumzerlegung $(T = (I, F), \{X_i \mid i \in I\})$ der Breite k zu einem Graphen haben, so können wir diese in polynomieller Zeit in folgende Normalform bringen, ohne ihre Breite zu verändern:

- *T ist ein binärer Baum (d.h. alle inneren Knoten haben Verzweigungsgrad 1 oder 2)*
- *Hat ein Knoten $i \in I$ zwei Söhne j und k, so gilt $X_i = X_j = X_k$*
- *Hat ein Knoten $i \in I$ nur einen Sohn j, so unterscheiden sich X_i und X_j um genau ein Element ($X_i = X_j \cup \{v\}$ oder $X_j = X_i \cup \{v\}$)*

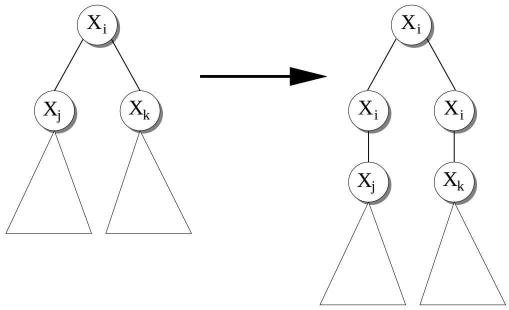
Beweis: Jeder Baum läßt sich in einen Binärbaum umwandeln, indem man Knoten geeignet dupliziert:



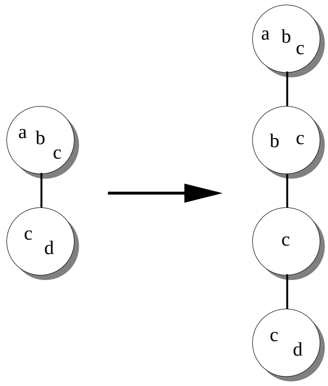
Das können wir in polynomieller Zeit erledigen und der Baum wird auch höchstens polynomiell größer dabei.

Hat ein Knoten i zwei Söhne j und k, ändern wir den Baum wie folgt, so daß die beiden Söhne von i die gleichen Elemente enthalten:

[5]Die Laufzeit des Algorithmus ist exponentiell in k.



Gibt es nun noch benachbarte Knoten, die sich um mehr als ein Element unterscheiden, so fügen wir zwischen den beiden Knoten eine Kette neuer Knoten ein, die sich jeweils nur um ein Element unterscheiden.



Auch dies geht alles in polynomieller Zeit, ändert die Breite der Zerlegung nicht, und macht den Baum allenfalls polynomiell größer. ■

3.5 Planar TMAX und Planar TMIN

Wir konstruieren nun die PTAS-Algorithmen für unsere drei Probleme. Dabei fangen wir mit den Problemen Planar TMAX und Planar TMIN an, da die Algorithmen hier etwas einfacher (und damit durchschaubarer) sind. Zunächst zeigen wir, daß die Probleme für Eingaben, deren Abhängigkeitsgraph p-außerplanar ist, in polynomieller Zeit optimal lösbar sind.

Anschließend werden wir dieses Resultat verwenden, um einen **PTAS** für beliebige 'planare' Eingaben zu konstruieren.

3.5.1 Lösung auf p-außerplanaren Graphen

Lemma 3.15
Sei p eine feste Zahl. Dann sind Planar TMAX *und* Planar TMIN *für Eingaben $\mathcal{I}$, bei denen der Abhängigkeitsgraph $G_{\mathcal{I}}$ p-außerplanar ist, in* **PO**.

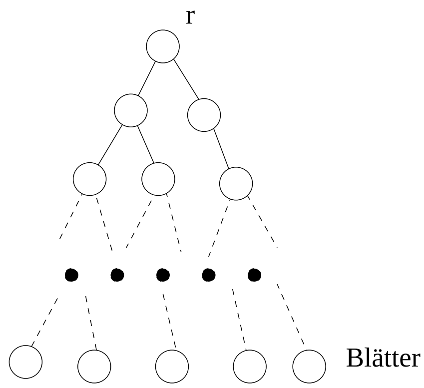


Abbildung 3.6: So sind Bäume bei uns orientiert: die Wurzel r oben, die Blätter unten

Beweis: Sei eine Eingabe $\mathcal{I} = (\{\varphi_1, \ldots, \varphi_n\}, \{x_1, \ldots, x_m\}, w)$ gegeben, deren Abhängigkeitsgraph $G_\mathcal{I}$ p-außerplanar ist. Nach Satz 3.13 können wir in polynomieller Zeit eine Baumzerlegung $(T = (I, F), \{X_i \mid i \in I\})$ von $G_\mathcal{I}$ berechnen, die Breite $\leq 3p - 1$ hat, und oBdA in Normalform vorliegt (Lemma 3.14).

Sei r die Wurzel des Baumes T, den wir uns mit der Wurzel oben und den Blättern unten vorstellen (Abbildung 3.6).

Für jeden Knoten $i \in I$ des Baumes sei

- $T[i]$ der Teilbaum von T, der i als Wurzel hat[6],
- $F[i]$ die Menge der Formeln φ_j, die im Baum $T[i]$ vorkommen, und
- $V[i]$ die Menge der Variablen x_j, die im Baum $T[i]$ vorkommen.

Wir lösen das Problem, eine optimale Variablenbelegung zu finden, mittels dynamischer Programmierung von unten nach oben. Wir beginnen bei den Blättern und setzen wiederholt die Teillösungen zusammen, bis wir an der Wurzel angekommen sind.

Das Array $best_i$

Für jeden Knoten i mit $X_i = \{\varphi_{a_1}, \ldots, \varphi_{a_r}, x_{b_1}, \ldots, x_{b_s}\}$, berechnen wir dazu ein Array $best_i$. Dieses enthält einen Eintrag für jedes Tupel $(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1},$

[6]Insbesondere also $T[r] = T$ und für Blätter b besteht $T[b]$ nur aus b.

$\ldots, v_{b_s})$, wobei ψ_{a_j} jeweils ein Implikant von φ_{a_j} ist, und die $v_{b_j} \in \{0,1\}$ sind.[7]

Dabei enthält $best_i(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ den maximalen (TMAX) bzw. minimalen (TMIN) Wert $\sum_{x_j \in V[i], \mathfrak{I}(x_j)=1} w(x_j)$ unter allen Belegungen $\mathfrak{I}$ mit

- $\mathfrak{I} \models \psi_{a_j}$ für $j = 1, \ldots, r$,
- $\mathfrak{I}(x_{b_j}) = v_{b_j}$ für $j = 1, \ldots, s$,
- $\mathfrak{I}$ erfüllt alle Formeln in $F[i]$.

Falls es keine solche Belegung gibt, setzen wir $best_i(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) := \infty$.

Bevor wir darstellen, wie diese Berechnung genau abläuft, machen wir uns klar, wie wir schließlich die Kosten der optimalen Lösung des Optimierungsproblems erhalten. Dies ist einfach der größte (für TMAX) bzw. kleinste (für TMIN) endliche Wert, den das Array $best_r$ an der Wurzel enthält.

Der im folgenden angegebene Algorithmus berechnet nur die Kosten der optimalen Lösung. Indem wir im Array *best* nicht nur die Kosten, sondern auch gleich eine passende optimale Belegung der Variablen speichern, können wir jedoch auch leicht eine optimale Lösung ermitteln. Der übersichtlicheren Darstellung wegen verzichten wir im folgenden darauf.

Berechnung von $best_i$ an den Blättern

Die Berechnung der Arrays $best_i$ erfolgt, wie bereits gesagt, von unten nach oben. Für Blätter $b \in I$ mit $X_b = \{\varphi_{a_1}, \ldots, \varphi_{a_r}, x_{b_1}, \ldots, x_{b_s}\}$ gibt es für den Wert von $best_b(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ zwei Möglichkeiten:

1. Es gibt keine passende Belegung. Das passiert genau dann, wenn ein x_{b_j} mit $v_{b_j} = 0$ positiv in einem ψ_{a_k} vorkommt, oder ein x_{b_j} mit $v_{b_j} = 1$ negativ in einem ψ_{a_k} vorkommt. Wir nennen dann das Tupel $(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ **inkonsistent**.

   In diesem Falle setzen wir den Wert auf ∞.

2. Andernfalls (das Tupel ist **konsistent**) gibt es zumindest eine passende Belegung, nämlich diejenige, die den x_{b_j} jeweils den Wert v_{b_j} zuordnet, und den restlichen Variablen den Wert 0 (für TMAX) bzw. 1 (für TMIN).

   Hier setzen wir den Wert auf $\sum_{j=1}^{s} v_{b_j} \cdot w(x_{b_j})$.

[7] Da die Anzahl der Implikanten durch die Eingabelänge N nach oben beschränkt ist, hat das Array höchstens die Größe $O(N^r 2^s) = O(N^{r+s}) = O(N^k)$, d.h. es ist polynomiell groß.

Berechnung von $best_i$ an den inneren Knoten

Betrachten wir nun die Berechnung von $best_i$ für einen inneren Knoten i unter der Voraussetzung, daß die Arrays $best_j$ des Sohnes bzw. der Söhne bereits berechnet wurden. Wir unterscheiden dabei drei Fälle:

i **hat zwei Söhne** j **und** k**:** Da der Baum in Normalform vorliegt, gilt dann $X_i = X_j = X_k$. Wir setzen:

$$\begin{aligned} best_i(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) &:= \\ best_j(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) &+ \\ best_k(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) - \sum_{l=1}^{s} v_{b_l} \cdot w(x_{b_l}) & \end{aligned}$$

Hierbei ist die Summe gleich ∞, wenn mindestens einer der Summanden gleich ∞ ist.

Daß diese Belegung 'richtig' im Sinne der Definition von $best_i$ auf Seite 32 ist, sieht man wie folgt. Seien $\mathfrak{I}_j$ und $\mathfrak{I}_k$ zu $(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ passende, optimale (im Sinne der Definition von $best_j$ und $best_k$) Belegungen für die Knoten j und k. Dann erhalten wir eine Belegung $\mathfrak{I}_i$ mit den angegebenen Kosten für den Knoten i wie folgt:

$$\mathfrak{I}_i(x) := \begin{cases} 1, & \text{falls } x \notin V[i] \text{ (für TMAX)}, \\ 0, & \text{falls } x \notin V[i] \text{ (für TMIN)}, \\ v_{b_l}, & \text{falls } x = x_{b_l}, \\ \mathfrak{I}_j(x), & \text{falls } x \in V[j] - V[k], \\ \mathfrak{I}_k(x), & \text{falls } x \in V[k] - V[j] \end{cases}$$

Das wichtigste hier ist, daß beim 'Verschmelzen' die letzten beiden Fälle (in der obigen Fallunterscheidung) wirklich unabhängig voneinander sind. Denn die Variablen in $V[j]$, die nicht in X_i vorkommen, tauchen auch in $V[k]$ nicht auf (und umgekehrt).

i **hat einen Sohn** j**, und es gilt** $X_i \subsetneq X_j$**:** Dann enthält X_j nach Definition genau ein Element mehr als X_i.

Gilt $X_j = X_i \cup \{\varphi_l\}$, so setzen wir im Falle von TMAX:

$$\begin{aligned} best_i(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) &:= \\ \max_{\psi_l \text{ Implikant von } \varphi_l} best_j(\psi_{a_1}, \ldots, \psi_{a_r}, \psi_l, v_{b_1}, \ldots, v_{b_s}) & \end{aligned}$$

Gilt $X_j = X_i \cup \{x_l\}$, so setzen wir im Falle von TMAX:

$$best_i(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) := \max_{v_l \in \{0,1\}} best_j(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}, v_l)$$

Die 'max' laufen dabei nur über die $best_j(\ldots)$ mit Wert $\neq \infty$. Sind alle der $best_j(\ldots)$ unendlich, so setzen wir auch $best_i$ auf ∞.

Im Falle von TMIN benutzen wir jeweils 'min' statt 'max'.

Der Definition von $best_i$ wird genüge getan, da wir nur weniger fordern (wir vergessen die Belegung von ψ_l bzw. x_l).

i hat einen Sohn j, und es gilt $X_i \supsetneq X_j$: Gilt $X_i = X_j \cup \{\varphi_l\}$, so setzen wir für alle Implikanten ψ_l von φ_l:

$$best_i(\psi_{a_1}, \ldots, \psi_{a_r}, \psi_l, v_{b_1}, \ldots, v_{b_s}) := \begin{cases} \infty, \text{falls Tupel inkonsistent,} \\ best_j(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}), \text{sonst} \end{cases}$$

Falls $X_i = X_j \cup \{x_l\}$, so setzen wir für $v_l \in \{0,1\}$:

$$best_i(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}, v_l) := \begin{cases} \infty, \text{falls Tupel inkonsistent,} \\ best_j(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) \\ \qquad\qquad + v_l \cdot w(x_l), \text{sonst} \end{cases}$$

Auch hier sieht man leicht, daß die Definition von $best_i$ weiterhin erfüllt ist. ■

3.5.2 Zusammensetzen für Planar TMAX

Kommen wir nun zum Fall, daß der Abhängigkeitsgraph $G_{\mathcal{I}}$ unserer Eingabe $\mathcal{I}$ ein beliebiger planarer Graph ist. Wir betrachten zunächst das Problem Planar TMAX. Das Vorgehen im folgenden entspricht dem in Abbildung 3.3 schematisch dargestellten Ablauf.

Zur Lösung berechnen wir zunächst eine Einbettung des Abhängigkeitsgraphen in die Ebene, was in polynomieller Zeit machbar ist (wie z.B. schon in [HoTa74] angedeutet). Die Menge der in der Einbettung außenliegenden Knoten bezeichnen wir mit L_1 (siehe Abbildung 3.2, Seite 26). Die Knoten, die außen liegen, wenn wir den Knoten von L_1 entfernen, bezeichnen wir mit L_2, usw. Allgemein ist L_i die Menge der außenliegenden Knoten, wenn wir die Knoten aus $L_1, \ldots, L_{i-1}$ entfernen. Der Graph $G_{\mathcal{I}}$ wird somit in eine Reihe von Schichten $L_1, L_2, \ldots, L_t$ zerlegt (Abbildung 3.2).

Sei $p := \lceil \frac{1}{\varepsilon} \rceil$. Um das Problem auf den $O(1)$-außerplanaren Fall zu reduzieren, setzen wir alle Variablen in den Schichten L_i mit $i \equiv 0, 1 \mod 2p$ gleich 0 und ersetzen sie auch in den Formeln durch den Ausdruck 'falsch'. Dadurch erhalten wir ein neues Problem $\mathcal{I}'$. In dessen Schichten $(L_i)_{i \equiv 0,1 \mod 2p}$ kommen nur noch Formeln φ_j vor, und diese sind nicht durch Kanten verbunden. Deshalb zerfällt der Abhängigkeitsgraph $G_{\mathcal{I}'}$ in lauter $2p$-außerplanare Teile $P_i = \bigcup_{j=1}^{2p} L_{2pi+j}$.

Da diese Teile unabhängig voreinander sind (sie haben keine Formeln oder Variablen gemeinsam), können wir mit dem bereits bekannten Algorithmus für jedes Teilproblem P_i in polynomieller Zeit eine optimale Belegung $\mathfrak{I}_i$ berechnen. Eine Lösung des ursprünglichen Problems $\mathcal{I}$ erhalten wir durch Zusammenfügen dieser Einzellösungen:

$$\mathfrak{I}(x_k) := \begin{cases} \mathfrak{I}_i(x_k), & \text{falls } x_k \text{ in } P_i \text{ vorkommt,} \\ 0, & \text{sonst } (x_k \text{ liegt in einer Schicht } L_i \text{ mit } i \equiv 0, 1 \mod 2p) \end{cases}$$

Die Belegung $\mathfrak{I}$ erfüllt alle Formeln der Eingabe $\mathcal{I}$, da alle Variablen in den Schichten $(L_i)_{i \equiv 0,1 \mod 2p}$ auf 0 gesetzt wurden und die $\mathfrak{I}_i$ die restlichen Variablen nach Voraussetzung so belegen, daß die Formeln erfüllt werden.

Wie gut ist diese Belegung im Vergleich zu einer optimalen Belegung $\mathfrak{I}_{opt}$? Da $\mathfrak{I}$ auf jedem P_i optimal ist, ist das Gesamtgewicht der hier mit 1 belegten Variablen mindestens so groß wie bei $\mathfrak{I}_{opt}$. Da $\mathfrak{I}$ jedoch alle Variablen in den Schichten L_i mit $i \equiv 0, 1 \mod 2p$ mit 0 belegt, gilt:

$$cost(\mathfrak{I}) \geq cost(\mathfrak{I}_{opt}) - \delta_0, \text{wobei} \tag{3.1}$$

$$\delta_0 := \sum_{x_j \in L_i, i \equiv 0,1 \mod 2p} w(x_j) \cdot \mathfrak{I}_{opt}(x_j)$$

Das garantiert uns noch nicht, daß $\mathfrak{I}$ eine ε-approximative Lösung ist, denn $\mathfrak{I}_{opt}$ kann sehr viele Variablen in den $(L_i)_{i \equiv 0,1 \mod 2p}$ mit 1 belegen, δ_0 also sehr groß sein.

Nun war es aber völlig willkürlich, gerade die Variablen in den Schichten $(L_i)_{i \equiv 0,1 \mod 2p}$ auf 0 zu setzen, wir hätten genauso die Variablen in den Schichten $(L_i)_{i \equiv 2k, 2k+1 \mod 2p}$ für ein festes $k \in \{0, \ldots, p-1\}$ wählen können. Wenn wir dies für alle möglichen k tun, erhalten wir insgesamt p Lösungen $\mathfrak{I}^k$ mit

$$cost(\mathfrak{I}^k) \geq cost(\mathfrak{I}_{opt}) - \delta_k, \text{wobei}$$

$$\delta_k := \sum_{x_j \in L_i, i \equiv 2k, 2k+1 \mod 2p} w(x_j) \cdot \mathfrak{I}_{opt}(x_j)$$

Offenbar gilt $\sum_{i=0}^{p-1} \delta_i = cost(\mathfrak{I}_{opt})$, nach dem Schubfachprinzip gibt es also mindestens ein $k \in \{0, \ldots, p-1\}$ mit $\delta_k \leq \frac{1}{p} cost(\mathfrak{I}_{opt})$ und somit

$$cost(\mathfrak{I}^k) \geq cost(\mathfrak{I}_{opt}) - \frac{1}{p} cost(\mathfrak{I}_{opt}) \geq (1-\varepsilon) \cdot cost(\mathfrak{I}_{opt})$$

Die Lösung mit den höchsten Kosten $\mathfrak{I}^k$ ist also eine ε-approximative Lösung. Da die Laufzeit zwar von p, p aber nur von ε abhängt, haben wir einen PTAS.

3.5.3 Zusammensetzen für Planar TMIN

Für Planar TMIN benutzen wir dasselbe Prinzip des Zerlegens in $O(1)$-außerplanare Graphen, Berechnen der optimalen Lösung auf den Teilen und Zusammensetzen der Teillösungen zu einer Gesamtlösung. Die Details der Zerlegung und des Zusammensetzens sind jedoch anders als bei TMAX.

Sei wieder $p := \lceil \frac{1}{\varepsilon} \rceil$. Die Teilprobleme P_i $(0 \leq i < \frac{t}{2p})$ bestimmen wir so:

P_i besteht aus allen Formeln φ_j, die in den Schichten $L_{2pi+1}, L_{2pi+2}, \ldots, L_{2pi+2p}$ liegen (und den Variablen, die in diesen Formeln vorkommen).

Der Abhängigkeitsgraph dieser Teilprobleme ist offenbar höchstens $(2p+2)$-außerplanar,[8] sie lassen sich also in polynomieller Zeit optimal lösen. Weiterhin kommt wegen $p \geq 1$ jede Variable in höchstens 2 Teilproblemen vor.

Durch Lösen der Teilprobleme P_i erhalten wir Belegungen $\mathfrak{I}_i$ für die Variablen in P_i. Wir fügen diese wie folgt zu einer Belegung $\mathfrak{I}$ aller Variablen zusammen:

$$\mathfrak{I}(x_j) = \begin{cases} \mathfrak{I}_i(x_j), & \text{falls } x_j \text{ nur in } P_i \text{ vorkommt,} \\ \mathfrak{I}_i(x_j), & \text{falls } x_j \text{ in } P_i \text{ und } P_{i+1} \text{ vorkommt, und } \mathfrak{I}_i(x_j) = \mathfrak{I}_{i+1}(x_j), \\ 1, & \text{sonst} \end{cases}$$

Die Belegung $\mathfrak{I}$ erfüllt alle Formeln. Denn liegt eine Formel φ_k in einem Teilproblem P_i, so wird die Formel durch die Belegung $\mathfrak{I}_i$ erfüllt. In $\mathfrak{I}$ sind allenfalls *mehr* Variablen auf 1 gesetzt. Da Variablen nur positiv (wir betrachten TMIN!) in φ_k vorkommen, wird φ_k also auch von $\mathfrak{I}$ erfüllt.

Wie gut ist die so erhaltene Lösung? Sei $\mathfrak{I}_{opt}$ eine optimale Belegung aller Variablen. Da die $\mathfrak{I}_i$ optimal für jedes Teilproblem P_i sind, gilt:

$$\sum_{x_j \in P_i} \mathfrak{I}_i(x_j) \cdot w(x_j) \leq \sum_{x_j \in P_i} \mathfrak{I}_{opt}(x_j) \cdot w(x_j)$$

[8] Da die zu den Formeln eines Teilproblems gehörigen Variablen noch in den angrenzenden Schichten L_{2pi} und $L_{2pi+2p+1}$ liegen können, erhalten wir das “+2”.

Durch Summieren über alle Teilprobleme erhalten wir (man beachte, daß dabei Variablen in den Schichten L_i mit $i \equiv 0,1 \mod 2p$ möglicherweise doppelt gezählt werden):

$$cost(\mathfrak{I}) \leq \sum_i \sum_{x_j \in P_i} \mathfrak{I}_i(x_j) \cdot w(x_j) \leq \sum_i \sum_{x_j \in P_i} \mathfrak{I}_{opt}(x_j) \cdot w(x_j) \leq cost(\mathfrak{I}_{opt}) + \delta_0,$$

$$\text{wobei } \delta_0 := \sum_{x_j \in L_i, i \equiv 0,1 \mod 2p} \mathfrak{I}_{opt}(x_j) \cdot w(x_j)$$

Diese Ungleichung entspricht der Ungleichung (3.1) für TMAX. Der restliche Beweis läuft dann auch analog: man unterteilt nicht nur bei den Schichten $(L_i)_{i \equiv 0,1 \mod 2p}$, sondern bei allen möglichen $(L_i)_{i \equiv 2k, 2k+1 \mod 2p}$, und die beste der so erhaltenen p Lösungen ist dann ε-approximativ. ■

3.6 Planar MPSAT

Der im vorherigen Abschnitt angegebene Algorithmus für Planar TMAX und TMIN läßt sich für Planar MPSAT im wesentlichen übernehmen. Wir gehen daher nur auf die Veränderungen ein. Sei $K := \sum_{i=1}^{n} w(\varphi_i)$. Der Algorithmus, den wir zunächst angeben, hat eine Laufzeit polynomiell in N (der Größe der Eingabe) und K. Nach Angabe des Algorithmus zeigen wir, wie der Algorithmus durch Runden der niedrigstwertigen Bits in ein PTAS umgewandelt werden kann.

3.6.1 Lösung auf p-außerplanaren Graphen

Lemma 3.16
Sei p eine feste Zahl. Dann ist Planar MPSAT *für Eingaben $\mathcal{I}$, bei denen $G_\mathcal{I}$ p-außerplanar ist, in* **FPTAS**.

Beweis:

Das Array $best_i$

Wiederum berechnen wir 'von unten nach oben' ein Array $best_i$. Sei $X_i = \{\varphi_{a_1}, \ldots, \varphi_{a_r}, x_{b_1}, \ldots, x_{b_s}\}$. Dann betrachten wir als Indizes Tupel der Form $(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ mit

- c **Zahl aus** $\{0, 1, \ldots, K\}$

- ψ_{a_j} entweder Implikant von φ_{a_j} **oder gleich** NULL[9]
- $v_{b_j} \in \{0,1\}$

Für den Inhalt der Arrayelemente gilt diesmal: $best_i(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ hat den minimalen Wert $\sum_{x_j \in V[i], \mathfrak{I}(x_j)=1} w(x_j)$ unter allen Belegungen $\mathfrak{I}$ mit

- $\mathfrak{I} \models \psi_{a_j}$ für alle $j = 1, \ldots, r$ mit $\psi_{a_j} \neq \text{NULL}$,
- $\mathfrak{I}(x_{b_j}) = v_{b_j}$ für $j = 1, \ldots, s$,
- $\sum_{\varphi_j \in F[i], \mathfrak{I} \models \varphi_j} w(\varphi_j) = c$.

Falls es keine solche Belegung gibt, setzen wir $best_i(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ gleich ∞.

Wir nennen ein Tupel $(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ konsistent, falls keine durch das Tupel mit 0 belegte Variable positiv, und keine mit 1 belegte Variable negativ in einem ψ_{a_j} vorkommt. In einem ψ_{a_j} =NULL kommt keine Variable vor.

Berechnung von $best_i$ an den Blättern

Die Berechnung beginnt wiederum an den Blättern. Für jedes Blatt b setzen wir für alle konsistenten Tupel $(\psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$:

$$best_b \left(\sum_{1 \leq j \leq r, \psi_{a_j} \neq \text{NULL}} w(\varphi_{a_j}), \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s} \right) := \sum_{j=1}^{s} v_{b_j} \cdot w(x_{b_j})$$

Für alle anderen Tupel setzen wir den Wert von $best_b$ auf ∞.

Berechnung von $best_i$ an den inneren Knoten

Beim Zusammenfügen an einem inneren Knoten i unterscheiden wir wieder drei Fälle:

i **hat zwei Söhne** j **und** k**:** Es gilt $X_i = X_j = X_k$. Für jedes der Tupel $(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})$ setzen wir

$$c' := \sum_{1 \leq j \leq r, \psi_{a_j} \neq \text{NULL}} w(\varphi_{a_j}), \text{ und}$$

[9] Der Wert NULL soll dabei bedeuten, daß die betreffende Formel durch die Belegung nicht (unbedingt) erfüllt wird.

$$\begin{aligned} best_i(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) := \\ \min_{d \in \{c', c'+1, \ldots, c\}} (best_j(d, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) \\ + best_k(c + c' - d, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s})) \\ - \sum_{l=1}^{s} v_{b_l} \cdot w(x_{b_l}) \end{aligned}$$

i **hat einen Sohn** j**, und es gilt** $X_i \subsetneq X_j$ Das verläuft sinngemäß genau wie bei TMAX/TMIN: wir nehmen zu einem Tupel von $best_i$ die 'beste' Erweiterung aus $best_j$, d.h. die mit dem kleinsten $best_j$-Wert.

i **hat einen Sohn** j**, und es gilt** $X_i \supsetneq X_j$ Falls $X_i = X_j \cup \{\varphi_l\}$, so setzen wir für alle Implikanten ψ_l von φ_l und für $\psi_l =$ NULL:

$$best_i(c, \psi_{a_1}, \ldots, \psi_{a_r}, \psi_l, v_{b_1}, \ldots, v_{b_s}) :=$$

$$\begin{cases} best_j(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}), & \text{falls } \psi_l = \text{ NULL}, \\ best_j(c - w(\varphi_l), \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}), & \text{falls Tupel konsistent} \\ & \text{und } c \geq w(\varphi_l), \\ \infty, & \text{sonst} \end{cases}$$

Falls $X_i = X_j \cup \{x_l\}$, so setzen wir für $v_l \in \{0, 1\}$:

$$best_i(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}, v_l) :=$$

$$\begin{cases} best_j(c, \psi_{a_1}, \ldots, \psi_{a_r}, v_{b_1}, \ldots, v_{b_s}) + v_l \cdot w(x_l), & \text{falls Tupel konsistent,} \\ \infty, & \text{falls Tupel inkonsistent} \end{cases}$$

In allen drei Fällen sieht man leicht, daß die $best_i$ definierende Eigenschaft erfüllt bleibt.

Die Laufzeit des Algorithmus ist polynomiell in N (der Problemgröße) und K (!). Durch Weglassen der niedrigstwertigen Bits der $w(\varphi_i)$ erhält man einen **FPTAS**. Auf die Details gehen wir am Ende des nächsten Abschnitts ein. ■

3.6.2 Zusammensetzen

Sei nun eine beliebige Eingabe mit planarem Abhängigkeitsgraphen gegeben. Wie schon bei PLANAR TMAX/TMIN setzen wir $p := \lceil \frac{1}{\varepsilon} \rceil$.

Diesmal zerlegen wir das Problem, indem wir alle *Formeln* in den Schichten L_i mit $i \equiv 2k, 2k+1 \mod 2p$ aus der Eingabe entfernen (wobei k eine feste Zahl aus $\{0, \ldots, p-1\}$ ist). Die resultierenden Teilprobleme haben dann $2p$-außerplanare Abhängigkeitsgraphen, für die wir optimale Lösungen in polynomieller Zeit berechnen können. Durch geeignete Wahl von k können wir analog zu PLANAR TMAX/TMIN garantieren, daß die im folgenden berechnete Lösung ε-approximativ ist.

Für jedes Teilproblem P_i wurde ein *best*-Array an der Wurzel berechnet. Aus diesem können wir ein Array $c_i(0 \ldots K)$ ermitteln, bei dem $c_i(j)$ angibt, was das Mindestgewicht an mit 1 belegten Variablen ist, um einen Wert der Kostenfunktion von j zu erreichen.

Nun interessiert uns ein Array $c(0 \ldots K)$, das dieselbe Information für das gesamte Problem enthält. Der höchste besetzte Eintrag mit Wert $\leq b$ ist dann der ε-approximative Wert der optimalen Kosten. Wir berechnen c mittels dynamischer Programmierung:

1. `c := c`$_0$
2. `FOR i := 1 TO` $\frac{t}{2p}$
   (a) `tmp := c`
   (b) `FOR j := 0 TO` K

$$\texttt{c(j)} := \min_{0 \leq k,l \leq K, k+l=j} \texttt{tmp(k)} + \texttt{c}_i\texttt{(l)}$$

Die Korrektheit dieses Algorithmus sieht man ein, wenn man folgende Invariante im Auge behält. Nach dem i-ten Durchlauf der äußeren Schleife enthält $c(j)$ für das kleinste Variablengewicht, mit dem man auf den ersten i Teilproblemen einen Wert der Kostenfunktion von j erreichen kann.

3.6.3 Runden der $w(\varphi_i)$

Wie schon der Algorithmus für p-außerplanare Graphen hat der soeben vorgestellte den Nachteil, in K polynomielle Laufzeit zu besitzen. Ähnlich wie im vorangehenden Kapitel bei **MAX-DP** behelfen wir uns mit dem Trick, die niedrigstwertigen Bits der Formelgewichte $w(\varphi_i)$ auf 0 zu setzen. Dabei wollen wir nun abschätzen, wie sich der Wert der optimalen Lösung verändert, wenn man die Kostenfunktion auf diese Weise ändert.

Dabei können wir oBdA davon ausgehen, daß alle φ_i durch Variablenbelegungen mit einem Gewicht $\leq b$ erfüllbar sind, d.h. einen Implikanten enthalten, dessen Variablen zusammen ein Gewicht $\leq b$ haben. Denn falls dies für eine Formel nicht gilt, können wir sie aus der Eingabe entfernen, ohne das Ergebnis zu ändern.

Seien also n Formeln mit Gesamtgewicht K gegeben. Nach dem Schubfachprinzip muß es mindestens eine erfüllbare Formel mit Gewicht $\frac{1}{n}K$ geben. Wir setzen $q := \lceil \log\left(\frac{\varepsilon}{n^2}K\right)\rceil$, und gehen über zur Kostenfunktion w' mit $w'(\varphi_i) := 2^q \left\lfloor \frac{w(\varphi_i)}{2^q} \right\rfloor$ und $w'(x_i) := w(x_i)$.

Die Anzahl der möglichen Werte der ersten Komponente unserer $best_i$ Arrays, als auch der Indizes der c_i Arrays ist dann polynomial in n und $\frac{1}{\varepsilon}$:

$$O\left(\frac{K}{2^q}\right) = O\left(\frac{K}{\frac{\varepsilon}{n^2}K}\right) = O\left(\frac{n^2}{\varepsilon}\right)$$

Bei festem ε, somit also festem p, ist die Laufzeit damit polynomial in N, was zu zeigen war.

Der Unterschied zwischen den optimalen Lösungen für die Kostenfunktionen w und w' läßt sich wie folgt abschätzen (man vergleiche mit der Rechnung auf Seite 18). Hierbei sind *opt* die Kosten der optimalen Lösung für die Kostenfunktion w, *opt'* dasselbe für w' und $\mathfrak{I}_{opt}$ eine optimale Belegung für w.

$$\begin{aligned}\text{Fehler} = \frac{opt - opt'}{opt} &\leq \frac{\sum_{\mathfrak{I}_{opt}\models\varphi_i} w(\varphi_i) - \sum_{\mathfrak{I}_{opt}\models\varphi_i}(w(\varphi_i) - 2^q)}{\sum_{\mathfrak{I}_{opt}\models\varphi_i} w(\varphi_i)} \\ &= \frac{\sum_{\mathfrak{I}_{opt}\models\varphi_i} 2^q}{\sum_{\mathfrak{I}_{opt}\models\varphi_i} w(\varphi_i)} \leq \frac{n\cdot 2^q}{\frac{1}{n}K} = \frac{n^2\cdot 2^q}{K} \leq \frac{n^2\cdot 2\frac{\varepsilon}{n^2}K}{K} = 2\varepsilon\end{aligned}$$

Was passiert, wenn wir diese Rundung in unseren im vorherigen Abschnitt beschriebenen Algorithmus einbauen? Durch das Runden erhalten wir ein Problem, dessen optimale Lösung $(1-2\varepsilon)$ der Kosten ursprünglich besten Lösung hat. Unser Algorithmus findet also ein Lösung mit mindestens $(1-2\varepsilon)(1-\varepsilon) = 1 - 3\varepsilon + 2\varepsilon^2 \geq 1 - 3\varepsilon$ der Kosten der optimalen Lösung des ursprünglichen Problems. Wir haben somit einen PTAS. ■

Kapitel 4

Diskussion

In dieser Arbeit haben wir Problemklassen und Probleme betrachtet, die vielleicht erste Schritte hin zur syntaktischen Charakterisierung der Komplexitätsklassen **FPTAS** und **PTAS** sein könnten.

Wie weit sind wir von diesem Ziel noch entfernt?

4.1 FPTAS

Kann mit der Klasse **MAX-DP** eine syntaktische Charakterisierung der Komplexitätsklasse **FPTAS** gelingen? Sind die Probleme aus **MAX-DP** gute Repräsentanten der Probleme aus **FPTAS**? Die letzte Frage ist nicht einfach zu beantworten, da erstaunlich wenige natürliche Probleme aus **FPTAS** bekannt sind[1], das bekannteste ist Knapsack (und Varianten davon).

Zwei Eigenschaften der Klasse **MAX-DP** sind besonders bemerkenswert.

1. Die lineare Kostenfunktion. Sie ist offenbar entscheidende Voraussetzung für die Anwendbarkeit der dynamischen Programmierung, die es ermöglicht, den Lösungsraum in nur nM statt M^n (seiner eigentlichen Größe!) Schritten nach der optimalen Lösung zu durchsuchen.

   Ebenfalls eine Folge der linearen Kostenfunktion ist es, daß sich die Werte der Lösungsfunktion $s(1), s(2), \ldots, s(n)$ nacheinander wählen lassen, da sie unabhängig voneinander in die Kostenfunktion eingehen.

   Sind lineare Kostenfunktionen charakteristisch für die Klasse **FPTAS**?

2. Die Lösungsraumstruktur. Um eine approximative Lösung zu erhalten, setzten wir die niedrigstwertigen Bits der Funktionswerte auf 0,

[1] Immerhin haben wir mit der Klasse **MAX-DP** nun eine reichhaltige Quelle unnatürlicher Probleme aus **FPTAS** :-)

betrachteten also statt jeder einzelnen möglichen Lösungsfunktion nur noch jede m-te, wobei m von ε abhing. Wir durchliefen den Lösungsraum also mit einer größeren Schrittweite bzw. auf einem groberen Raster.

Wenn der Lösungsraum sehr unstrukturiert wäre, mit ständigem auf und ab der Kostenfunktion, dann wäre dieses Vorgehen zum Finden einer approximativen Lösung wohl hoffnungslos. Bei **MAX-DP** klappt das Verfahren jedoch hervorragend.

Ist diese Lösungsraumstruktur, bei der sich die 'Schrittweite' mit der man den Raum durchsucht und die Qualität der daraus resultierenden Lösung so genau entsprechen, typisch für **FPTAS**? Findet man bei jedem **FPTAS**-Problem bessere Lösungen, indem man die Suche 'verfeinert'?

Auf den ersten Blick erscheint das fraglich. Denn aus Sicht eines **FPTAS**-Algorithmus bedeutet ein kleiner werdendes ε zunächst nur, daß mehr Rechenzeit zur Verfügung steht. Was der Algorithmus mit dieser Zeit anstellt, ist völlig offen, und es scheint keinen Grund zu geben, warum das etwas mit Verfeinerung der Suche, oder Verändern der Schrittweite zu tun haben sollte. Auch scheint es nicht zwingend, daß die Kostenfunktion linear von der Lösung abhängt.

Aber vielleicht verstehen wir die Klasse **FPTAS** nur noch nicht gut genug? Ein besseres Verständnis der Klasse **FPTAS** scheint zwingend zu sein, um bei dem Problem der syntaktischen Charakterisierung weitere Fortschritte zu machen.

4.2 PTAS

Wie steht es mit **PTAS**? Die interessanteste gemeinsame Eigenschaft unserer drei Probleme scheint ihre 'Zerlegbarkeit' zu sein.

Alle Lösungsalgorithmen benutzten die Tatsache, daß sich die Probleme in Teilprobleme zerlegen ließen, die weitgehend unabhängig voneinander waren. Und das sowohl bei der Zerlegung in $O(1)$-außerplanare Graphen, als auch bei der Lösung der Teilprobleme auf den $O(1)$-außerplanaren Graphen. Im letzteren Fall war entscheidend, daß man das Problem so zerlegen konnte, daß man immer nur konstant viele Variablen und Formeln auf einmal betrachten mußte.

Sehr wichtig war die Forderung nach der Planarität der Abhängigkeitsgraphen. Eine Variable kommt dann nämlich nur in Formeln 'in ihrer Nähe' vor, und weit entfernte Variablen tauchen nicht in denselben Formeln auf.

Änderungen in der Variablenbelegung haben demnach auch nur lokale Folgen. Ist das typisch für **PTAS**-Probleme?

Interessanterweise scheinen sich einige Probleme, für die erst kürzlich die Zugehörigkeit zu **PTAS** gezeigt wurde, nicht ohne weiteres auf unsere drei Probleme reduzieren zu lassen. So etwa Euclidian TSP [Arora96], der Fall von TSP, bei dem die n Orte Punkte in der Ebene sind, und die Abstandsfunktion c gerade dem euklidischen Abstand zwischen den Punkten entspricht. Oder auch Weighted Planar Graph TSP [AGKKW98], bei dem die Eingabe ein gewichteter planarer Graph ist, auf dem die Rundreise vollführt werden muß.

Beide Probleme operieren ebenfalls auf 'planaren' Eingaben (allerdings nicht planar im Sinne von Kapitel 3), und auch bei diesen Problemen ist eine geschickte Zerlegung der Eingabe der wesentliche 'Trick' zur Lösung des Problems.

Vielleicht wäre es ein guter nächster Schritt, die Gemeinsamkeiten der verschiedenen Methoden, Probleme in Teilprobleme zu zerlegen, zu untersuchen. Möglicherweise lernt man damit mehr über **PTAS** und kommt einer syntaktischen Charakterisierung näher.

Literaturverzeichnis

[AbVi89] S. Abiteboul, V. Vianu, *Fixpoint extensions of first-order logic and Datalog-like languages*, Proceedings LICS'89, 71–79, 1989.

[AGKKW98] S. Arora, M. Grigni, D. Karger, P. Klein, A. Woloszyn, *A Polynomial-Time Approximation Scheme for Weighted Planar Graph TSP*, Proceedings SIAM SODA, 1998.

[ALMSS92] S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, *Proof Verification and Hardness of Approximation Problems*, Proceedings STOC'92, 14–23, 1992.

[Arora96] S. Arora, *Polynomial-time Approximation Schemes for Euclidean TSP and other Geometric Problems*, Proceedings FOCS'96, 2–13, 1996.

[Baker83] B.S. Baker, *Approximation Algoritms for NP-Complete Problems on Planar Graphs*, Proceedings FOCS'83, Arizona, 1983.

Eine ausführlichere Version findet sich in [Baker94].

[Baker94] B.S. Baker, *Approximation Algoritms for NP-Complete Problems on Planar Graphs*, Journal of the ACM, 41:153–180, 1994.

[Bodlaender88] H.L. Bodlaender, *Some Classes of Graphs with Bounded Treewidth*, Bulletin of the European Association for Theoretical Computer Science, 36:116–126, 1988.

[Bodlaender93a] H.L. Bodlaender, *A Linear Time Algorithm for Finding Tree-decompositions of Small Treewidth*, Proceedings STOC'93, 226–234, 1993.

[Bodlaender93b] H.L. Bodlaender, *A Tourist Guide through Treewidth*, Acta Cybernetica, 11:1–21, 1993.

[CrKa95] P. Crescenzi, V. Kann, *A compendium of NP optimization problems*, Technical report SI/RR-95/02, Universita di Roma "La Sapienza", 1995.

[EbFl95] H.D. Ebbinghaus, J. Flum, *Finite Model Theory*, Springer Verlag, 1995.

[EFT96] H.D. Ebbinghaus, J. Flum, W. Thomas, *Einführung in die mathematische Logik*, Spektrum Akademischer Verlag, 1996.

[Fagin74] R. Fagin, *Generalized first-order spectra and polynomial-time recognizable sets*, in: R.M. Karp (ed.), Complexity of Computation, SIAM-AMS Proceedings, 7:43–73, 1974.

[GaJo79] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[GoWi95] M.X. Goemans, D.P. Williamson, *Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming*, Journal of the ACM, 42:1115–1145, 1995.

[Håstad97] J. Håstad, *Some optimal inapproximability results*, Proceedings STOC'97, 1–10, 1997.

[HoTa74] J. Hopcroft, R.E. Tarjan, *Efficient planarity testing*, Journal of the ACM, 21:549–568, 1974.

[Immerman86] N. Immerman, *Relational queries computable in polynomial time*, Information and Control, 68:86–104, 1986.

[Immerman87] N. Immerman, *Expressibility as a complexity measure: results and directions*, Second Structure in Complexity Conference, 194–202, 1987.

[Khanna96] S. Khanna, *A Structural View of Approxiation*, Dissertation, Stanford University, 1996.

[KhMo96] S. Khanna, R. Motwani, *Towards a Syntactic Characterization of PTAS*, Proceedings STOC'96, 329–337, 1996.

[KMSV94] S. Khanna, R. Motwani, M. Sudan, U. Vazirani, *On Syntactic versus Computational Views of Approximability*, Proceedings FOCS'94, 819–830, 1994.

[KST97] S. Khanna, M. Sudan, L. Trevisan, *Constraint satisfaction: The approximability of minimization problems*, Proceedings 12th Annual IEEE Conference on Computational Complexity, 1997.

[KSW97] S. Khanna, M. Sudan, D.P. Williamson, *A complete classification of the approximability of maximization problems derived from Boolean constraint satisfaction*, Proceedings STOC'97, 11–20, 1997.

[Malmström96a] A. Malmström, *Logic and Approximation*, Dissertation, RWTH Aachen, 1996.

[Malmström96b] A. Malmström, *Optimization problems with approximation schemes*, Proceedings CSL'96, Volume 1258 of LNCS, 316–333, 1997.

[Papadimitriou94] C.H. Papadimitriou, *Computational Complexity*, Addison Wesley, 1994.

[PaYa91] C.H. Papadimitriou, M. Yannakakis, *Optimization, Approximation, and Complexity Classes*, Journal of Computer and System Sciences, 43:425–440, 1991.

[RoSe83] N. Robertson, P. D. Seymour, *Graph minors I. Excluding a forest*, J. Comb. Theory Series B, 35:39–61, 1983.

[RoSe86] N. Robertson, P. D. Seymour, *Graph minors II. Algorithmic aspects of tree-width*, J. Algorithms, 7:309–322, 1986.

[SaGo76] S. Sahni, T. Gonzalez, *P-complete approxmiation problems*, Journal of the ACM, 23:555–565, 1976.

[Schöning97] U. Schöning, *Theoretische Informatik – Kurz gefaßt*, Spektrum Akademischer Verlag, 1997.

[Vardi82] M.Y. Vardi, *The complexity of relational query languages*, Proceedings STOC'82, 137–146, 1982.