

Approximationsalgorithmen

Wintersemester 2006

Marco Lübbecke

Technische Universität Berlin
Institut für Mathematik, Sekr. MA 6-1
Straße des 17. Juni 136
D-10623 Berlin
m.luebbecke@math.tu-berlin.de

15. Februar 2007

Inhaltsverzeichnis

1	Vorbereitungen	3
1.1	Definitionen und Vereinbarungen	3
1.2	Untere Schranken für <i>OPT</i>	4
1.3	Welche Fragen stellen wir?	6
2	Grenzen der Approximierbarkeit	7
2.1	Nicht-Approximierbarkeit	7
2.2	Greedy-Algorithmus für Set Cover	8
2.3	Konstante Approximation	9
2.4	Polynomiale Approximationsschemata	11
2.5	Absolute Approximation	15
3	LP-Basierte Methoden	16
3.1	Lineare Programme, Dualität, Optimalität	16
3.2	Runden der fraktionalen Lösung	18
3.2.1	Deterministisch	18
3.2.2	Randomisiert	19
3.3	Dual Fitting	21
3.4	Ganz- und Halbzahligkeit	25
3.4.1	Weighted Vertex Cover	26
4	Primal-Duales Schema	28
4.1	Komplementärer Schlupf relaxiert	28
4.2	Grundversion Primal-duales Schema	29
4.3	Minimum Multicut auf Bäumen	33
4.4	Primal-Duales Schema mit Synchronisierung	36
4.5	Allgemeine Anwendung im Netzwerk Design	38
5	Aroras PTAS für Euklidisches TSP	44
5.1	Portale und Portaltouren	45
5.2	Dynamisches Programm	47
5.3	Randomisiert verschobene Partitionierung	49
6	Semidefinite Relaxation	53
7	Facility Location und k-Median	59
7.1	Metrisches unkapazitiertes Facility Location	59

8 Iteriertes Runden	63
8.1 Survivable Network Design	63
9 Maximum Satisfiability	70
10 Hardness of Approximation	77
10.1 PCP Theorem	78
10.2 Die Klasse Max \mathcal{SNP}	81
11 Über den Tellerrand...	85
Literatur	88

Kapitel 1

Vorbereitungen

last edit:
18.10.06

Wir wissen, dass wir für \mathcal{NP} -schwere Optimierungsprobleme keinen polynomialen Algorithmus finden können, der eine optimale Lösung für alle Instanzen findet, wenn die weithin akzeptierte Gleichung $\mathcal{P} \neq \mathcal{NP}$ richtig ist. In diesem Sinne sind alle schweren Probleme gleich schwer. Die Theorie der Approximationsalgorithmen erlaubt es uns, diese „Schwierigkeit“ wesentlich besser zu differenzieren. Es wird sich herausstellen, dass einige Probleme „beliebig gut“ zu approximieren sind, während es für andere genauso schwer ist eine approximative Lösung zu bestimmen wie das Problem optimal zu lösen!

Die Lineare Programmierung [10] wird fast wie ein roter Faden ein häufig wiederkehrendes Thema in unseren Algorithmen sein und eine grundlegende Kenntnis der Dualität linearer Programme wird sich als hilfreich erweisen. Der Stoff der Vorlesung basiert zu großen Teilen auf den Büchern von Hochbaum [22] und Vazirani [36]. Weiterhin sehr interessant in unserem Kontext sind der (\mathcal{NP} -Vollständigkeits-)Klassiker [12] und das jüngere Kompendium [7], das auch online¹ verfügbar ist. Weiterführende Literatur ist darüberhinaus jeweils angegeben.

1.1 Definitionen und Vereinbarungen

Definition 1.1 Ein Optimierungsproblem Π ist charakterisiert durch ein 4-Tupel:

1. Eine Menge Π von Probleminstanzen (die wir mit dem Problem selbst identifizieren)
2. Zu jeder Instanz $I \in \Pi$ gehört eine Menge $Lsg(I)$ zulässiger Lösungen
3. Jede Lösung $y \in Lsg(I)$ hat einen (nicht-negativen, rationalen) Wert $v(I, y)$
4. max oder min

Den optimalen Wert einer Instanz bezeichnen wir mit $OPT(\Pi, I)$, oder kurz OPT , wenn der Zusammenhang klar ist.

Beispiel. In einem ungerichteten Graphen $G = (V, E)$ ist eine minimale *Knotenüberdeckung* (VERTEX COVER) $S \subseteq V$ zu finden. Das bedeutet, dass jede Kante $e \in E$ einen Endknoten in S haben muss, und $|S|$ soll so klein wie möglich sein. Dieses \mathcal{NP} -schwere Problem stellt sich nach unserer Definition ganz offenbar folgendermaßen als Optimierungsproblem dar:

¹<http://www.nada.kth.se/~viggo/problemlist/compendium.html>

1. $\Pi = \{G = (V, E) \mid G \text{ ist ein Graph}\}$
2. $Lsg(G) = \{S \subseteq V \mid \forall (u, v) \in E : u \in S \vee v \in S\}$
3. $v(G, S) = |S|$
4. min

Definition 1.2 Die Klasse \mathcal{NPO} der \mathcal{NP} -Optimierungsprobleme besteht aus Optimierungsproblemen Π , für die gilt:

1. Instanzen $I \in \Pi$ können in polynomialer Zeit erkannt werden
2. Es existiert ein Polynom q so, dass zu jeder Instanz $I \in \Pi$ und jeder Lösung $y \in Lsg(I)$ gilt: $|y| \leq q(|I|)$. Darüberhinaus können wir zu jedem y mit $|y| \leq q(|I|)$ in Polynomialzeit entscheiden, ob $y \in Lsg(I)$.
3. $v(I, y)$ ist für jedes $y \in Lsg(I)$ in Polynomialzeit berechenbar

Die Klasse \mathcal{NPO} erweitert in natürlicher Weise die Klasse \mathcal{NP} von Entscheidungsproblemen auf Optimierungsprobleme. Ist ein Optimierungsproblem in \mathcal{NPO} , so gehört das zugehörige Entscheidungsproblem zu \mathcal{NP} . Ein \mathcal{NPO} -Optimierungsproblem ist \mathcal{NP} -schwer, wenn das zugehörige Entscheidungsproblem \mathcal{NP} -vollständig ist, s. Ausiello et al. [7].

Definition 1.3 Ein α -Approximationsalgorithmus \mathcal{A} für ein Minimierungsproblem Π

1. berechnet zu jeder Instanz $I \in \Pi$ eine zulässige Lösung mit Wert $\mathcal{A}(I) \leq \alpha \cdot OPT(I)$
2. und hat eine Laufzeit polynomial in der Eingabegröße $|I|$.

In dieser Definition ist $\alpha \geq 1$ und für Maximierungsprobleme *mutatis mutandis* $\alpha \leq 1$. Wir nennen α den *Approximationsfaktor*, *-garantie*, *-güte* oder englisch *approximation ratio* oder *performance guarantee*. Wie aber können wir einen Algorithmus analysieren, wie eine Approximationsgarantie abgeben, wenn es \mathcal{NP} -schwer ist, OPT zu bestimmen? Auf eine gute Charakterisierung des Optimums (im Sinne eines min-max-Resultats wie etwa der LP-Dualität oder dem Max-Flow-Min-Cut Theorem) können wir nicht hoffen.

1.2 Untere Schranken für OPT

Wenn wir wissen, wie groß OPT mindestens ist, wenn wir also eine untere Schranke $LB(I) \leq OPT(I)$ kennen, können wir das ausnutzen. Tatsächlich ist eine gute untere Schranke häufig bereits ein beachtlicher Schritt zum Auffinden eines Approximationsalgorithmus. Wir schauen uns hierzu noch einmal das Problem VERTEX COVER an.

Wieviele Knoten brauchen wir mindestens für eine Überdeckung? Sofern $|E| \geq 1$ benötigen wir trivialerweise mindestens einen Knoten. Tatsächlich können wir versuchen, eine Menge von Kanten zu finden, die paarweise keinen Endknoten gemeinsam haben. Eine bestmögliche (also größte) solche Menge zu finden ist die Frage nach einem maximalen Matching M in

Algorithmus 1 Maximales Matching für VERTEX COVER

Bestimme ein maximales Matching M in G

Gib die Endknoten $S_{MM} = \{u, v \in V \mid (u, v) \in M\}$ der Kanten in M aus.

G . Ein solches maximales Matching kann man z.B. mit einem einfachen Greedy-Algorithmus bestimmen. *Jede* Knotenüberdeckung muss mindestens einen Endknoten jeder Kante $e \in M$ enthalten (sonst wäre e nicht überdeckt). Daraus folgern wir, dass $|M| \leq OPT$.

Satz 1.4 *Algorithmus 1 ist ein 2-Approximationsalgorithmus für VERTEX COVER.*

Beweis. Zunächst einmal ist S_{MM} eine Knotenüberdeckung von G : Gäbe es eine unüberdeckte Kante \bar{e} , wäre auch $M \cup \{\bar{e}\}$ ein Matching im Widerspruch zur Maximalität von M .

Da nach obiger Überlegung $|S_{MM}| = 2 \cdot |M| \leq 2 \cdot OPT$, folgt die Behauptung. \square

Dies ist die typische Weise, wie untere Schranken für OPT im Beweis der Approximationsgarantie eingesetzt werden. Aber war unsere *Analyse des Algorithmus* nun „scharf“ oder könnten wir auch eine bessere Güte garantieren? Das folgende Beispiel zeigt, dass die Analyse nicht zu verbessern ist.

Beispiel. Betrachte den vollständigen bipartiten Graphen $K_{n,n}$ mit $2n$ Knoten. Algorithmus 1 findet ein maximales (bipartites) Matching mit $|M| = n$, so dass $|S_{MM}| = 2n$. Eine minimale Knotenüberdeckung kommt aber mit $OPT = n$ Knoten (nämlich alle Knoten einer Partition) aus, und daher ist für diese Familie von Instanzen $|S_{MM}| = 2 \cdot OPT$. Der englische Begriff *tight example* ist hier am treffendsten. Wir werden stets verlangen, dass *tight examples* beliebig groß werden können: Instanzen beschränkter Größe könnten wir mittels vollständiger Enumeration in Polynomialzeit explizit optimal lösen.

Wir können uns nun etwas allgemeiner fragen, ob mit dieser konkreten unteren Schranke ein anderer Algorithmus gefunden werden kann, der eine bessere Approximationsgarantie aufweist. Die Frage ist also, ob wir die untere Schranke bestmöglich algorithmisch ausnutzen.

Beispiel. Sei K_n ein vollständiger Graph mit n ungerade. Die Kardinalität eines maximalen Matchings ist $LB = (n-1)/2$, während $OPT = n-1$ gilt. Weil $OPT/LB = 2$, können wir mit *dieser* unteren Schranke LB keinen besseren Approximationsfaktor beweisen. Diese Einsicht wird uns wieder begegnen, wenn wir untere Schranken aus linearen Programmen gewinnen.

Wir haken nun noch einmal nach und fragen, ob es einen Algorithmus geben kann, der eine *andere* untere Schranke verwendet und zu einer besseren Gütegarantie kommt. Und tatsächlich stellen wir damit eine natürliche Frage, auf die bisher niemand eine Antwort geben konnte, nämlich wie gut das Problem VERTEX COVER approximiert werden kann.

Offenes Problem 1.5 *Welches ist das kleinste α , für das es einen α -Approximationsalgorithmus für VERTEX COVER gibt?*

Es handelt sich hierbei ebenfalls um die Frage nach einer unteren Schranke, dieses Mal nach einer (scharfen) unteren Schranke der Approximierbarkeit eines *Problems*. Für VERTEX COVER ist das beste bekannte Resultat von Dinur und Safra [11]:

Satz 1.6 *Es ist \mathcal{NP} -schwer, VERTEX COVER mit einem Faktor besser als $10\sqrt{5} - 21$ ($\approx 1,36067$) zu approximieren.*

Die zurzeit beste bekannte Approximationsgüte ist übrigens $2 - \Theta(\frac{1}{\sqrt{\log n}})$ [26], also nur um $o(1)$ besser die Güte des trivialen Algorithmus 1. Es ist unklar, ob der Faktor $2 - \varepsilon$ für festes $\varepsilon > 0$ erreicht werden kann. Hochbaum vermutet in ihrem Buch, dies sei nicht möglich [22].

Zusammenfassend hier noch einmal eine Übersicht über die Werte, die uns bei der Analyse eines α -Approximationsalgorithmus interessieren (es sollte auch klar sein, warum uns eine untere Schranke vom Wert 0 in einer Analyse nichts bringt):

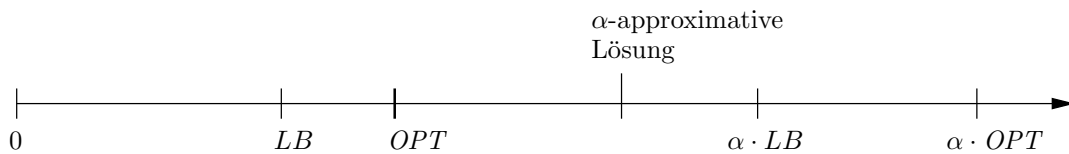


Abbildung 1.1: Schranken auf dem Zahlenstrahl

1.3 Welche Fragen stellen wir?

- Woher bekommt man (gute) untere Schranken für OPT ?
- Welche Approximation garantiert ein Algorithmus?
- Ist die Analyse des Algorithmus scharf?
- Welches ist eine untere Schranke für die Approximierbarkeit unseres Problems?

Bemerkung. Die Frage liegt nahe, wie *praxisrelevant* ein Approximationsalgorithmus sein kann, dessen Lösungen einen Wert höchstens um 100% über dem Optimum haben (oder warum man diese Güte auf z.B. 50% verbessern möchte). Wir werden sehen, dass genauso wie bei der Laufzeitanalyse von Algorithmen der *worst case* häufig so gezielt konstruiert und geradezu „pathologisch“ ist, dass wir solche Fälle in der Praxis fast ignorieren können. Implementationen von Approximationsalgorithmen erzielen im Allgemeinen wesentlich bessere Lösungen als der Approximationsfaktor vorhersagt. Auf der anderen Seite sind es gerade diese „schlimmen“ Beispiele, die uns die Intuition dafür geben, wo ein Algorithmus zu verbessern ist oder worin sich die unverrückbare Schwierigkeit eines Problems begründet.

Außerdem sei bemerkt: Die *Praxis* hat ohnehin noch viel weiterreichende, oft nicht-mathematische („schmutzige“) Forderungen. Sie interessiert uns in dieser Vorlesung nur nachrangig; vielmehr wollen wir uns mit der Schönheit der Algorithmen und der Mathematik der dahinter stehenden Analyse beschäftigen.

Bemerkung. Unsere Vorlesung beschäftigt sich nicht mit der garantierten Qualität einer Lösung unter der Annahme fehlerbehafteter Eingabedaten, auch nicht damit, dass reelle Zahlen im Computer durch rationale Zahlen angenähert dargestellt werden müssen, und auch nicht mit Rundungsfehlern während der Berechnungen. Dies (und noch viel mehr) ist Aufgabe der Numerik.

Kapitel 2

Grenzen der Approximierbarkeit

last edit:
24.10.06

Wie eingangs erwähnt, sind einige schwere Optimierungsprobleme in einem wohl-definierten Sinne schwerer als andere. Wir präzisieren diese Aussage, indem wir untere Schranken für die Approximierbarkeit eines Problems kennen lernen.

2.1 Nicht-Approximierbarkeit

Wir wissen, dass wir keinen polynomialen Algorithmus zur optimalen Lösung eines \mathcal{NP} -vollständigen Problems erwarten dürfen, es sei denn $\mathcal{P} = \mathcal{NP}$. Nach unserer Sprachregelung gibt es für diese Probleme keinen 1-Approximationsalgorithmus. Dies ist eine offensichtliche *untere Schranke* für die (Nicht-)Approximierbarkeit eines Problems. Es stellt sich die natürliche Frage, wie weit diese untere Schranke für ein konkretes Problem verbessert werden kann.

Satz 2.1 Sei $\alpha(n)$ eine in Polynomialzeit berechenbare Funktion. Wenn nicht $\mathcal{P} = \mathcal{NP}$, gibt es keinen $\alpha(n)$ -Approximationsalgorithmus für TSP mit n Städten.

Beweis. Sei \mathcal{A} ein α -Approximationsalgorithmus für TSP. Die Idee ist eine Reduktion von HAMILTONSCHER KREIS. Ein Graph $G = (V, E)$ mit n Knoten wird dabei in einen kantengewichteten, vollständigen Graphen G' mit n Knoten so überführt, dass

- besitzt G einen Hamiltonschen Kreis $\implies OPT(TSP, G') = n$
- besitzt G keinen Hamiltonschen Kreis $\implies OPT(TSP, G') > \alpha(n) \cdot n$

G' enthält alle Knoten von G . Das Gewicht einer Kante e in G' ist 1, falls $e \in E$ und $\alpha(n) \cdot n$, falls $e \notin E$. Genau dann, wenn G Hamiltonsch ist, gilt $OPT(TSP, G') = n$. Muss jede TSP-Tour eine Kante $e \notin E$ benutzen, gilt $OPT(TSP, G') > \alpha(n) \cdot n$. Weil \mathcal{A} ein $\alpha(n)$ -Approximationsalgorithmus ist, findet \mathcal{A} im ersten Fall eine Tour mit Wert höchstens $\alpha(n) \cdot n$ (also tatsächlich mit Wert genau n). \mathcal{A} kann also dazu benutzt werden, das \mathcal{NP} -vollständige Problem HAMILTONSCHER KREIS in Polynomialzeit zu entscheiden, ein Widerspruch. \square

Man verdeutliche sich die Tragweite dieses Resultats. Der nur scheinbare Widerspruch zur 3/2-Approximation von Christofides' Algorithmus löst sich auf, wenn man bedenkt, dass

dieser Algorithmus für den Spezialfall des METRISCHEN TSP gilt. Die Beweistechnik, eine *Lücke* zu erzeugen, sollte uns in guter Erinnerung bleiben.

MAX CLIQUE (also Bestimmen eines größten vollständigen Teilgraphen) ist zwar kein nicht-approximierbares Problem wie TSP, seine Approximierbarkeit ist aber ähnlich schwach. Ein beliebiger Knoten ist offenbar eine approximative Lösung mit Güte n . Mit Techniken aus Kapitel 10 kann man zeigen, dass diese triviale Garantie bestmöglich ist, d.h. dass ein Faktor $n^{1-\varepsilon}$ für festes $\varepsilon > 0$ nicht erreichbar ist, es sei denn $\mathcal{P} = \mathcal{NP}$.

2.2 Greedy-Algorithmus für Set Cover

Beim SET COVER Problem haben wir ein *Universum* U von n Elementen gegeben, eine Menge $\mathcal{S} = \{S_1, \dots, S_k\}$ von Teilmengen von U und eine Kostenfunktion $c : \mathcal{S} \rightarrow \mathbb{Q}_+$. Gesucht ist ein kostenminimales $\mathcal{C} \subseteq \mathcal{S}$, so das $\bigcup_{S \in \mathcal{C}} S$ jedes Element von U enthält („überdeckt“). Dieses Problem ist für die Theorie der Approximationsalgorithmen fundamental. Es wird uns als Veranschaulichung für algorithmische Ideen regelmäßig begegnen.

Möglicherweise die erste solcher Ideen, die man hat, ist eine Greedy-Strategie: Nehme iterativ eine Menge S in die Überdeckung auf, deren unüberdeckte Elemente im Durchschnitt am wenigsten kosten. Sei C die Menge der bereits überdeckten Elemente, so definieren wir die *Kosteneffektivität* von S als $c(S)/|S \setminus C|$. Der *Preis* $price(e)$ eines Elements e wird gesetzt als die Kosteneffektivität der Menge, die e zum ersten Mal überdeckt.

Algorithmus 2 Greedy-Algorithmus für SET COVER

```

 $C := \emptyset, \mathcal{C} := \emptyset$ 
while  $C \neq U$  do
  Finde eine kosteneffektivste Menge  $S$ 
  Setze  $price(e) = \frac{c(S)}{|S \setminus C|}$  für alle  $e \in S \setminus C$ 
  Nimm  $S$  in die Überdeckung  $\mathcal{C}$  auf und setze  $C \leftarrow C \cup S$ 
end while
Gib  $\mathcal{C}$  aus

```

Sei e_1, \dots, e_n eine Numerierung der Elemente in der Reihenfolge, wie sie vom Greedy-Algorithmus 2 überdeckt werden.

Lemma 2.2 Für jedes $k \in \{1, \dots, n\}$ gilt $price(e_k) \leq OPT/(n - k + 1)$.

Beweis. Die (noch nicht in \mathcal{C} aufgenommenen) Mengen einer optimale Überdeckung können in jeder Iteration die Elemente in $\overline{C} := U \setminus C$ zu Kosten höchstens OPT überdecken. Eine dieser Mengen muss daher eine Kosteneffektivität von höchstens OPT/\overline{C} haben. Am Anfang der Iteration, in der e_k überdeckt wurde, enthielt \overline{C} mindestens $n - k + 1$ Elemente. Weil e_k von der in dieser Iteration kosteneffektivsten Menge überdeckt wurde, folgt

$$price(e_k) \leq \frac{OPT}{\overline{C}} \leq \frac{OPT}{n - k + 1} .$$

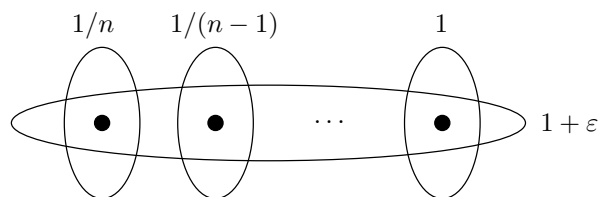
□

Satz 2.3 Der Greedy-Algorithmus ist ein H_n -Approximationsalgorithmus für SET COVER, wobei $H_n = 1 + \frac{1}{2} + \dots + \frac{1}{n}$.

Beweis. Die Kosten $c(S)$ jeder Menge $S \in \mathcal{C}$ verteilen sich gleichmäßig auf die von S zum ersten Mal überdeckten Elemente. Die Kosten der Überdeckung \mathcal{C} sind also $\sum_{k=1}^n \text{price}(e_k)$. Das ist nach Lemma 2.2 höchstens $H_n \cdot \text{OPT}$. \square

H_n heißt die n -te harmonische Zahl; es gilt $\ln(n+1) \leq H_n \leq 1 + \ln n$, d.h. H_n wächst wie der natürliche Logarithmus.

Beispiel. Hier ist ein *tight example* für den Greedy-Algorithmus:



Greedy nimmt (von links nach rechts) die n einelementigen Mengen, denn in jeder Iteration ist eine dieser Mengen am kosteneffektivsten. Die Kosten der Greedy-Überdeckung sind daher $\frac{1}{n} + \frac{1}{n-1} + \dots + 1 = H_n$, während $\text{OPT} = 1 + \varepsilon$. Man könnte nun meinen, dass sich eine so künstliche Panne durch ein geschickteres Verfahren vermeiden ließe. Alon, Moshkovitz und Safra [1] haben aber bewiesen, dass interessanterweise der naive Greedy-Ansatz das beste ist, was wir für SET COVER tun können.

Satz 2.4 Es gibt eine Konstante c so, dass SET COVER nicht besser als $c \cdot \ln n$ approximiert werden kann, es sei denn $P = \mathcal{NP}$.

Wir können dieses Resultat mit unseren Mitteln nicht beweisen. Viele Probleme begegnen uns in der Gestalt eines (Spezialfalls des) SET COVER Problems mit der „offensichtlichen“ $O(\log n)$ -Approximation, nämlich dem Greedy-Algorithmus. Die spannende Frage ist, ob der betrachtete Spezialfall eine bessere Approximation zulässt. Zum Beispiel ist VERTEX COVER ein spezielles SET COVER Problem. Greedy wählt einen Knoten v größten Grades, löscht v und alle zu v inzidenten Kanten, und iteriert mit dem verbleibenden Graph.

Übung. Gib einen Graph an, bei dem der Greedy-Algorithmus für VERTEX COVER nur eine Überdeckung mit Kosten $\Theta(\log n) \cdot \text{OPT}$ findet.

Die Leser mögen sich vielleicht gefragt haben, wo die vorher angepriesene Benutzung einer unteren Schranke für die Analyse des Greedy-Algorithmus versteckt war. Eine Formulierung als lineares Programm wird uns demnächst hierauf eine Antwort geben.

2.3 Konstante Approximation

\mathcal{NPO} -Probleme, die einen konstanten Approximationsfaktor erlauben, haben wir schon gesehen: METRISCHES TSP oder VERTEX COVER.

Definition 2.5 \mathcal{APX} ist die Klasse der \mathcal{NPO} -Probleme, für die ein α -Approximationsalgorithmus mit konstantem α existiert.

Wir wissen bereits, dass $\mathcal{APX} \subsetneq \mathcal{NPO}$: Etwa aus Satz 2.1 können wir folgern: Wenn TSP $\in \mathcal{APX}$, dann gilt $\mathcal{P} = \mathcal{NP}$. Auch haben wir gerade gesehen, dass SET COVER $\notin \mathcal{APX}$.

BIN PACKING ist das Problem, n Gegenstände der Größen $a_1, \dots, a_n \in (0, 1]$ in minimal wenige Bins (Dosen) der Größe 1 zu packen. *First Fit* ist ein einfacher 2-Approximationsalgorithmus: Anfangs ist kein Bin benutzt („geöffnet“). Die Gegenstände werden in beliebiger Reihenfolge in die bereits geöffneten Bins verteilt, jeweils in das erste passende. Passt der jeweils betrachtete Gegenstand in kein offenes Bin, wird ein neues geöffnet.

Für eine von *First Fit* erzeugte Packung mit v_{FF} Bins gilt, dass wenigstens $v_{FF} - 1$ davon halbvoll sind, d.h. $\sum_{i=1}^n a_i > (v_{FF} - 1)/2$. Die Summe aller a_i ist eine untere Schranke für OPT , daher gilt $v_{FF} - 1 < 2OPT \iff v_{FF} \leq 2OPT$, weil $v_{FF} \in \mathbb{N}$. Man kann tatsächlich einen 3/2-Approximationsalgorithmus angeben und das ist auch schon bestmöglich.

Satz 2.6 Sei Π ein Minimierungsproblem mit ganzzahligen Zielfunktionswerten und $k \in \mathbb{N}$ fest gewählt. Wenn es \mathcal{NP} -schwer ist zu entscheiden, ob $OPT \leq k$ dann gibt es keinen α -Approximationsalgorithmus mit $\alpha < 1 + 1/k$, es sei denn $\mathcal{P} = \mathcal{NP}$.

Beweis. Sei A ein α -Approximationsalgorithmus für Π mit $\alpha < (k + 1)/k = 1 + 1/k$. Algorithmus A berechnet in Polynomialzeit für *Ja*-Instanzen von Π eine Lösung mit Wert echt kleiner als $OPT \cdot (k + 1)/k \leq k \cdot (k + 1)/k = k + 1$ und damit tatsächlich mit Wert höchstens k . Für *Nein*-Instanzen hat jede Lösung offensichtlich einen Wert von wenigstens $k + 1$. Damit entscheidet A das \mathcal{NP} -schwere Problem ob $OPT \leq k$ in Polynomialzeit, ein Widerspruch zu $\mathcal{P} \neq \mathcal{NP}$. \square

Diesen „Standardtrick“ (der manchmal in \mathcal{NP} -Vollständigkeitsbeweisen „unbeabsichtigt“ ein Nebenresultat liefert) auf BIN PACKING angewendet bedeutet, dass es keinen Approximationsalgorithmus mit Faktor echt kleiner als 3/2 geben kann: Das Problem mit 2 Bins enthält als Spezialfall das \mathcal{NP} -schwere Problem PARTITION. Dieses Argument basiert auf kleinen Optimalwerten (hier: Anzahl Bins) trotz unbeschränkter Anzahl Gegenstände. Typischer wäre die Betrachtung von Optimalwerten, die mit n wachsen.

Der Algorithmus *Best-Fit Decreasing* packt die Gegenstände in der Reihenfolge nicht-aufsteigender Größe (die großen Gegenstände gelten als schwieriger zu packen). Außerdem wird jeder Gegenstand in ein offenes Bin gepackt, bei dem der freie Platz am besten verbraucht wird. Interessanterweise gilt für den Wert einer so gefundenen Packung $v_{BFD} \leq 11/9 OPT + 4$ [25], d.h. der Algorithmus hat einen Approximationsfaktor $11/9 + 4/OPT$. Man sagt, der *asymptotische Approximationsfaktor* ist 11/9, und damit offensichtlich kleiner als 3/2.

Bemerkung. Nicht bei jedem Problem in \mathcal{APX} lässt sich der Approximationsfaktor asymptotisch verbessern. Eine KNOTENFÄRBUNG eines Graphen $G = (V, E)$ (VERTEX COLORING, GRAPH COLORING) ist die Zuweisung $\chi : V \rightarrow \mathbb{N}$ einer Farbe zu jedem Knoten so, dass keine zwei adjazenten Knoten dieselbe Farbe erhalten: $\forall (u, v) \in E : \chi(u) \neq \chi(v)$. Die Anzahl der Farben ist zu minimieren (das Minimum nennt man die *chromatische Zahl* $\chi(G)$). Zu

last edit:
25.10.06

entscheiden, ob $\chi \leq 3$ ist \mathcal{NP} -vollständig; nach Satz 2.6 ist daher ein Approximationsfaktor besser als $4/3$ nicht möglich.

Dies ändert sich auch für wachsendes OPT nicht. Wir konstruieren dazu einen Graphen G_k wie folgt. G wird k -mal kopiert. Zusätzlich wird eine Kante zwischen je zwei Knoten eingefügt, wenn sie in verschiedenen Kopien sind (jede Kopie muss daher mit einer eigenen Farbmenge gefärbt werden); da für jede Kopie $\chi(G)$ Farben benötigt werden ist $\chi(G_k) \geq k \cdot \chi(G)$. Andererseits kann aus einer m -Färbung von G sofort eine km -Färbung von G_k erzeugt werden. Daher ist $\chi(G_k) \leq k \cdot \chi(G)$. Die benutzte Technik nennt man *Inflation*. Sie „bläst die $4/3$ -Lücke zu einer $4k/3k$ -Lücke auf“: Eine approximative Färbung für G_k mit echt weniger als $4/3 \cdot \chi(G_k)$ Farben muss nach Schubfachprinzip auf einer Kopie weniger als $4/3 \cdot \chi(G)$ Farben benutzen. Wir könnten damit $\chi(G) \leq 3$ in Polynomialzeit entscheiden. Dies macht einen asymptotischen Approximationsfaktor besser als $4/3$ unmöglich.

2.4 Polynomiale Approximationsschemata

In der Klasse \mathcal{APX} geht es lediglich um die Existenz *eines* konstanten Approximationsfaktors; nicht darum, dass dieser beliebig klein gewählt werden kann. Diese Forderung ist stärker.

Definition 2.7 *Ein polynomiales Approximationsschema (polynomial time approximation scheme, PTAS) für Problem Π ist eine Familie $\{A_\varepsilon\}$ von Algorithmen für Π so, dass für jedes $\varepsilon > 0$ gilt: A_ε ist ein $(1 + \varepsilon)$ -Approximationsalgorithmus für Π .*

Für gegebenes (weil fest gewähltes) $\varepsilon > 0$ kann die Laufzeit von A_ε allerdings exponentiell in ε sein. In der folgenden Definition fordern wir mehr.

Definition 2.8 *Eine Familie $\{A_\varepsilon\}$ von Algorithmen für Problem Π heißt voll polynomiales Approximationsschema (fully polynomial time approximation scheme, FPTAS, FPAS) für Π , falls für jedes $\varepsilon > 0$ gilt: A_ε ist ein $(1 + \varepsilon)$ -Approximationsalgorithmus für Π und die Laufzeit von A_ε ist polynomial auch in $\frac{1}{\varepsilon}$.*

Weil mit verschwindendem ε die Laufzeit von A_ε gegen unendlich geht, ergibt sich ein Wert für ε , ab dem vollständige Enumeration effizienter ist als ein Approximationsschema. Nur wenige uns bekannte Probleme erlauben Approximationsschemata und man kann diesen Teilklassen von \mathcal{NPO} (tatsächlich von \mathcal{APX}) wiederum Namen geben (sinnigerweise **PTAS** und **FPTAS**). Es gilt **FPTAS** \subseteq **PTAS** \subseteq \mathcal{APX} . Die Inklusionen sind strikt, wenn nicht $\mathcal{P} = \mathcal{NP}$, s. etwa [7].

„Standardmäßig“ entwickelt man ein Approximationsschema so, dass man je nach geforderter Approximationsgüte die auftretenden Zahlen einer Instanz herunterskaliert, somit kontrolliert an Genauigkeit verliert, dafür aber polynomiale Laufzeit für die skalierte Instanz erhält und letztlich durch vollständige Enumeration löst. Das FPTAS für KNAPSACK ist ein Beispiel für diese Technik.

Asymptotische Polynomiale Approximationsschemata

Als Konsequenz von Satz 2.6 kann es für BIN PACKING kein PTAS geben. Tatsächlich können wir den Begriff des PTAS wie beim asymptotischen Approximationsfaktor abschwächen und erhalten folgenden Satz.

Satz 2.9 Für jedes $0 < \varepsilon \leq 1/2$ gibt es einen Algorithmus A_ε mit Laufzeit polynomial in n , der für BIN PACKING eine Lösung mit höchstens $(1 + 2\varepsilon)OPT + 1$ Bins berechnet.

Die Familie $\{A_\varepsilon\}$ von Algorithmen dieses Satzes bildet ein *asymptotisches PTAS* für BIN PACKING, denn für jedes $\varepsilon > 0$ existiert ein $N > 0$ und ein Algorithmus A der Familie mit Approximationsfaktor $1 + \varepsilon$ für alle Instanzen mit $OPT \geq N$. Den Beweis von Satz 2.9 führen wir in zwei Zwischenschritten (Lemma 2.10 und 2.11).

Lemma 2.10 Seien $\varepsilon > 0$ und $K \in \mathbb{N}$ fest. BIN PACKING mit höchstens K verschiedenen Größen der Gegenstände und $a_1, \dots, a_n \geq \varepsilon$ lässt sich in Polynomialzeit optimal lösen.

Beweis. Keine Bin kann mehr als $\lfloor 1/\varepsilon \rfloor$ Gegenstände enthalten. Es gibt daher nicht mehr als $R := K^{\lfloor 1/\varepsilon \rfloor}$ verschiedene Möglichkeiten, eine Bin zu packen. Weiterhin werden nicht mehr als n Bins geöffnet, es gibt daher höchstens

$$\binom{n+R}{R} = \frac{(n+R)!}{n! \cdot R!} = \frac{(n+1)(n+2) \cdots (n+R)}{R!} \in O(n^R)$$

verschiedene zulässige Packungen (es ist $\binom{n+k-1}{n}$ die Anzahl Möglichkeiten, n identische Bälle in k Urnen zu verteilen). Weil R konstant ist, ist diese Anzahl polynomial. Man findet eine optimale Packung durch vollständige Enumeration in Polynomialzeit. \square

Lemma 2.11 Sei $\varepsilon > 0$ fest. Für BIN PACKING mit $a_1, \dots, a_n \geq \varepsilon$ gibt es einen $(1 + \varepsilon)$ -Approximationsalgorithmus.

last edit:
26.10.06

Beweis. Sei I die gegebene Instanz. Sortiere die n Gegenstände nicht-absteigend und partitioniere sie in $K := \lceil 1/\varepsilon^2 \rceil + 1$ Gruppen mit je höchstens $Q := \lfloor n\varepsilon^2 \rfloor$ Gegenständen. Da uns nur die Asymptotik interessiert, wählen wir n groß genug um $\varepsilon^2 Q \geq 1$ sicher zu stellen. Mit dieser Wahl können alle n Gegenstände wie angegeben gepackt werden.

Wir vergrößern jeden Gegenstand auf die Größe des größten in der jeweiligen Gruppe („aufrunden“). Diese aufgerundete Instanz nennen wir J und sie hat höchstens K verschiedene Größen der Gegenstände. Eine optimale Packung für J findet man mit Lemma 2.10. Diese Packung ist offensichtlich zulässig für die Originalgrößen der Instanz I .

Zur Abschätzung der Kosten von J erzeugen wir eine weitere Instanz J' , indem jeder Gegenstand auf die Größe des kleinsten Gegenstandes in seiner jeweiligen Gruppe verkleinert wird. Hier erhalten wir als untere Schranke $OPT(J') \leq OPT(I)$. Die entscheidende Beobachtung ist, dass aus jeder Packung für J' eine Packung für alle Gegenstände in J außer den höchstens Q Stück in der größten Gruppe konstruiert werden kann. Daher gilt

$$OPT(J) \leq OPT(J') + Q \leq OPT(I) + Q \leq (1 + \varepsilon)OPT(I)$$

Die letzte Ungleichung ergibt sich folgermaßen. Wegen $a_i \geq \varepsilon$, $i = 1, \dots, n$ gilt $n\varepsilon \leq OPT(I)$, also $Q = \lfloor n\varepsilon^2 \rfloor \leq \varepsilon OPT(I)$. Insgesamt ist J also $(1 + \varepsilon)$ -approximative Packung. \square

Beweis von Satz 2.9. Bezeichne mit I' die Instanz, die aus I durch Wegnehmen der Gegenstände mit $a_i < \varepsilon$ entsteht. Mit Lemma 2.11 finden wir eine Packung für I' mit höchstens

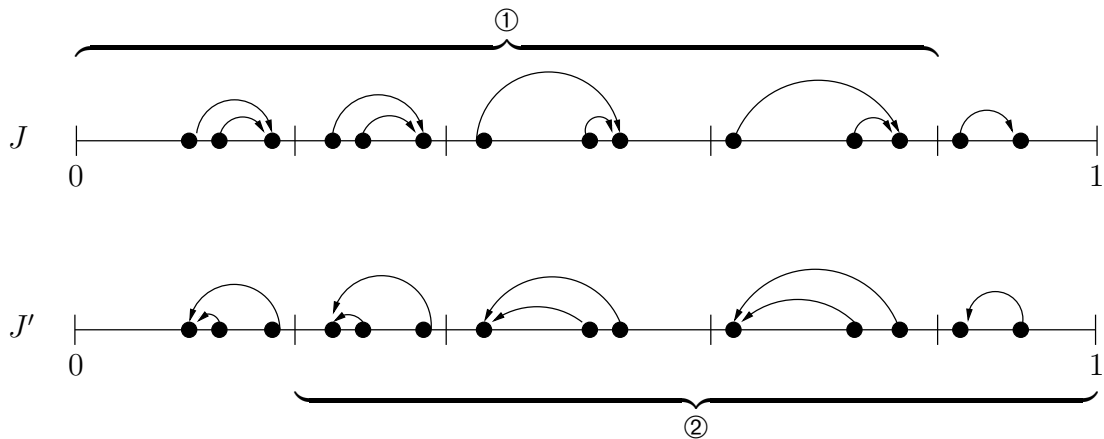


Abbildung 2.1: Zum Beweis von Lemma 2.11. Die Gegenstände in den Gruppen ② dominieren größtmäßig die Gegenstände in den Gruppen ①. Sollte die letzte Gruppe weniger als Q Gegenstände enthalten, so sind dennoch alle bis auf die größten Q Stück in J durch die Gegenstände in J' (bis auf die kleinsten Q Stück) dominiert.

$(1 + \varepsilon)OPT(I')$ Bins. Die verbleibenden Gegenstände mit $a_i < \varepsilon$ packen mit *First Fit* in die bereits geöffneten Bins, ggf. müssen hierbei neue Bins geöffnet werden. Falls nicht, benutzt die erhaltene Packung $(1 + \varepsilon)OPT(I') \leq (1 + \varepsilon)OPT(I)$ Bins.

Sei andernfalls M die Anzahl insgesamt geöffneter Bins. Weil wir mit *First Fit* weitere Bins geöffnet haben, sind alle Bins außer der letzten wenigstens zu $1 - \varepsilon$ gefüllt. Die Summe der Größen in I ist eine untere Schranke für $OPT(I)$, daher gilt

$$\begin{aligned} (M - 1)(1 - \varepsilon) &\leq OPT(I) \\ \iff M &\leq \frac{OPT(I)}{1 - \varepsilon} + 1 \\ &\leq (1 + 2\varepsilon)OPT(I) + 1 \end{aligned}$$

wobei in die letzte Ungleichung die Voraussetzung $\varepsilon \leq 1/2$ eingeht. □

Algorithmus 3 A_ε für BIN PACKING nach Satz 2.9

- Entferne alle Gegenstände kleiner als ε
 - Erzeuge durch Aufrunden konstante Anzahl verschiedener Größen (Lemma 2.10)
 - Bestimme für die gerundete Instanz eine $(1 + \varepsilon)$ -approximative Packung (Lemma 2.11)
 - Erzeuge aus dieser Packung eine Packung für die nicht gerundete Instanz
 - Packe alle Gegenstände kleiner als ε mit *First Fit*
-

Pseudopolynomialität

Für KNAPSACK gibt es einen $O(nP)$ -Algorithmus, wobei P eine obere Schranke für den optimalen Profit ist. Der Aufwand ist daher abhängig von der Größe der auftretenden Zahlen, die nicht polynomial in der Anzahl der Gegenstände beschränkt sein müssen. Ein anderes typisches „Zahlenproblem“ ist PARTITION. Bezeichne $|I|_\infty$ die größte in einer Instanz I auftretende Zahl. Wir definieren eine Abschwächung eines effizienten Algorithmus.

last edit:
31.10.06

Definition 2.12 Wir nennen einen Algorithmus pseudo-polynomial, wenn seine Laufzeitfunktion beschränkt ist durch ein Polynom in der Eingabegröße $|I|$ der Instanz und $|I|_\infty$.

Dies ist gleichbedeutend damit, dass ein pseudo-polynomialer Algorithmus noch polynomial in der *unären* (statt binären) Codierung einer Instanz sein darf. Beachte, dass die Laufzeit exponentiell in $\log |I|_\infty$ (der Länge der binären Codierung von $|I|_\infty$) ist, und somit die Existenz eines pseudo-polynomialen Algorithmus *nicht* $\mathcal{P} = \mathcal{NP}$ impliziert.

Satz 2.13 Sei p ein Polynom und Π ein \mathcal{NP} -schweres Minimierungsproblem so, dass $v(I, y)$ ganzzahlig für jedes $I \in \Pi$ und $y \in \text{Lsg}(I)$ und $\text{OPT}(\Pi, I) < p(|I|_\infty)$. Falls Π ein FPTAS besitzt, dann existiert ein pseudo-polynomialer Algorithmus für Π .

Beweis. Sei ein FPTAS für Π mit Laufzeit $q(|I|, 1/\varepsilon)$ auf Instanz I mit Fehler $\varepsilon > 0$ gegeben, wobei q ein Polynom sei. Setze $\varepsilon = 1/p(|I|_\infty)$ für Instanz I . Eine Lösung des FPTAS hat einen Wert kleiner als

$$(1 + \varepsilon)\text{OPT}(I) < \text{OPT}(I) + \varepsilon p(|I|_\infty) = \text{OPT}(I) + 1 .$$

Weil nur ganzzahlige Zielfunktionswerte auftreten, muss das FPTAS mit dem so definierten ε eine Optimallösung ausgeben. Die Laufzeit ist $q(|I|, p(|I|_\infty))$, also polynomial in $p(|I|_\infty)$. Dies ist ein pseudo-polynomialer Algorithmus für Π . \square

Definition 2.14 Wir nennen ein \mathcal{NP} -schweres Problem Π streng oder stark \mathcal{NP} -schwer oder \mathcal{NP} -schwer im strengen Sinne, wenn jedes Problem in \mathcal{NP} so auf Π reduziert werden kann, dass alle bei der Reduktion auftretenden Zahlen polynomial beschränkt bleiben.

Eine alternative Art, dies auszudrücken ist: Bezeichne mit Π_{poly} genau die Instanzen eines \mathcal{NP} -schweren Optimierungsproblems Π , bei denen $|I|_\infty$ polynomial in $|I|$ beschränkt ist. Wenn Π_{poly} \mathcal{NP} -schwer ist, dann ist Π streng \mathcal{NP} -schwer. Intuitiv sind für ein streng \mathcal{NP} -schweres Problem also *sogar* die Instanzen mit kleinen Zahlen schwer. Das gilt für die meisten Probleme, die uns begegnen. Insbesondere sind alle die \mathcal{NP} -schweren Probleme sogar streng \mathcal{NP} -schwer, die gar keine Zahlen in der Eingabe haben (wie HAMILTON PATH) oder bei denen die *sinnvollen* auftretenden Zahlen im Input natürlicherweise klein sind (z.B. ist $k \leq n$ in k -CLIQUE).

Wenn nicht $\mathcal{P} = \mathcal{NP}$, kann es für streng \mathcal{NP} -schwere Probleme keinen pseudo-polynomialen Algorithmus geben. Daher folgt aus Satz 2.13:

Korollar 2.15 Sei Π ein Problem mit den Voraussetzungen des Satz 2.13. Falls Π streng \mathcal{NP} -schwer ist, dann besitzt Π kein FPTAS, es sei denn $\mathcal{P} = \mathcal{NP}$.

Die nicht starken \mathcal{NP} -schweren Probleme nennen wir *schwach \mathcal{NP} -schwer*. Beispiele sind die oben genannten KNAPSACK und PARTITION. In Kapitel 10 werden wir die Klasse der Probleme kennen lernen, die kein PTAS besitzen. Auch für die neu definierten Komplexitätsklassen, am interessantesten wohl \mathcal{APX} , werden wir später Vollständigkeit definieren.

2.5 Absolute Approximation

Definition 2.16 Der absolute Fehler einer Lösung $y \in \text{Lsg}(I)$ einer Instanz I eines Optimierungsproblems Π ist definiert als $|\text{OPT} - v(I, y)|$.

Satz 2.17 Wenn nicht $\mathcal{P} = \mathcal{NP}$, gibt es keinen polynomialen Algorithmus für KNAPSACK mit absolutem Fehler.

Beweis. Sei X eine Menge von n Elementen mit Profiten p_1, \dots, p_n und Größen a_1, \dots, a_n , und sei b die Rucksackgröße. Wir nehmen an, das Problem erlaubt einen absoluten Fehler k . Wir modifizieren die Instanz, indem alle Profite mit $k + 1$ multipliziert werden, was die Menge der zulässigen Lösungen nicht ändert. Der Wert jeder Lösung der neuen Instanz ist nun ein Vielfaches von $k + 1$, und die einzige Lösung mit absolutem Fehler höchstens k ist die Optimallösung. Ein polynomialer Algorithmus mit absolutem Fehler k löst also das originale KNAPSACKPROBLEM in Polynomialzeit optimal, ein Widerspruch. \square

Das beste, was wir an Approximationsgüte erwarten können ist ein absoluter Fehler von 1 (Hochbaum nennt dies „wonderful“ [22]). Zwei Probleme mit dieser wunderbaren Güte folgen.

Knotenfärbung planarer Graphen VERTEX COLORING ist \mathcal{NP} -schwer selbst für planare Graphen. Der (nicht nur wegen seines Computerbeweises Aufsehen erregende) Vier-Farben-Satz besagt andererseits, dass jeder planare Graph in Polynomialzeit vierfärbbar ist.

Algorithmus 4 Knotenfärbung eines planaren Graphen G mit höchstens einer Extrafarbe

Falls G bipartit, berechne eine 2-Färbung
Ansonsten gib eine 4-Färbung aus.

Satz 2.18 Algorithmus 4 hat einen absoluten Fehler von höchstens 1.

Beweis. Offensichtlich, da $\text{OPT} \leq 4$. \square

Kantenfärbung einfacher Graphen (EDGE COLORING) Ein einfacher Graph mit Maximalgrad Δ kann nicht mit weniger als Δ Farben kantengefärbt werden. Der Satz von Vizing sagt andererseits, dass $\Delta + 1$ Farben ausreichen [35]. Der Beweis führt auf einen polynomialen Algorithmus, also ebenfalls einen Approximationsalgorithmus mit „wunderbarer“ Güte.

Am Ende dieses Kapitels ist uns klar, dass die Untersuchung der Approximierbarkeit wesentlich feinere Aussagen über die Schwierigkeit eines Problems zulässt als die einfache Aussage der \mathcal{NP} -Vollständigkeit. Die genaue Abgrenzung der Approximierbarkeit nach oben wie nach unten wird uns weiter beschäftigen.

Kapitel 3

LP-Basierte Methoden

last edit:
31.10.06

Die lineare Programmierung ist aus den Approximationsalgorithmen nicht mehr wegzudenken. Das schreckt eher theoretisch interessierte Studierende möglicherweise ab, aber zu Unrecht! Neben der Lösung linearer Programme als ausdrücklichem *Bestandteil* eines Approximationsalgorithmus, wird die zugrundeliegende Theorie als Werkzeug in der *Analyse* benutzt.

3.1 Lineare Programme, Dualität, Optimalität

Unter einem *linearen Programm* (LP) verstehen wir die Aufgabe,

$$z_{LP} = \min\{\mathbf{c}^T \mathbf{x} \mid A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

zu bestimmen. Wir nennen dies zur Verdeutlichung auch das *primale* lineare Programm. Es ist bekannt, dass sich Maximierungsprobleme, andere Restriktionen oder nicht vorzeichenbeschränkte Variablen auf diese Form zurückführen lassen. Das hierzu *duale lineare Programm* ist

$$z_{DP} = \max\{\mathbf{y}^T \mathbf{b} \mid \mathbf{y}^T A \leq \mathbf{c}^T, \mathbf{y} \geq \mathbf{0}\} .$$

Zur Wiederholung (und Abgrenzung der linearen Programmierung als „gut charakterisiertem“ und daher effizient lösbares Problem) hier die zentralen Resultate in Kurzform:

Satz 3.1 (Schwache Dualität) *Es gilt $z_{DP} \leq z_{LP}$.*

Satz 3.2 (Starke Dualität) *Es gilt $z_{DP} = z_{LP}$ genau dann, wenn das primale und das duale Programm zulässige Lösungen besitzen.*

Satz 3.3 (Satz vom Komplementären Schlupf)

Ein Paar primal bzw. dual zulässiger Lösungen $\mathbf{x} \in \mathbb{R}^n$ und $\mathbf{y} \in \mathbb{R}^m$ ist optimal \iff

Primale Bedingungen: Für $j = 1, \dots, n$ gilt: $x_j = 0$ oder $\sum_{i=1}^m a_{ij}y_i = c_j$.

Duale Bedingungen: Für $i = 1, \dots, m$ gilt: $y_i = 0$ oder $\sum_{j=1}^n a_{ij}x_j = b_i$.

Beweis. Nach schwacher Dualität gilt $\mathbf{y}^T \mathbf{b} \leq \mathbf{y}^T A \mathbf{x} \leq \mathbf{c}^T \mathbf{x}$; ist \mathbf{x}, \mathbf{y} ein optimales Paar, so gilt sogar Gleichheit. Umformung ergibt

$$\mathbf{y}^T (\mathbf{b} - A\mathbf{x}) = \mathbf{0} \quad \text{und} \quad (\mathbf{y}^T A - \mathbf{c}^T) \mathbf{x} = \mathbf{0} . \quad (3.1)$$

Der Beweis der Hinlänglichkeit folgt aus der komponentenweise zu lesenden Aussage, dass ein Produkt genau dann verschwindet, wenn einer seiner Faktoren verschwindet. Umgekehrt folgt aus (3.1), dass $\mathbf{y}^T \mathbf{b} = \mathbf{c}^T \mathbf{x}$. \square

Lineare Programme lassen sich mit der *Ellipsoid-Methode* in polynomialer Zeit lösen. Auf diesem Verfahren basiert das tiefe und folgenreiche Resultat der sogenannten *Äquivalenz zwischen Separierung und Optimierung* [18]. Es besagt, dass man auch solche linearen Programme in Polynomialzeit lösen kann, die eine exponentielle Anzahl von Restriktionen haben, ohne sie alle „hinschreiben“ zu müssen. Man muss hierfür einen *polynomialen* Algorithmus (manchmal: „Orakel“) angeben, der zu einer gegebenen Lösung entscheidet, ob sie zulässig ist, und wenn nicht, eine verletzte Restriktion ausgibt („Separierung“). Diese wird dem LP hinzugefügt und iteriert. Weil die Ellipsoid-Methode kein „kombinatorischer Algorithmus“ ist und man bei der Größe der auftretenden Zahlen in den Eingabedaten aufpassen muss, möchte man diese Argumente gerne nur spärlich einsetzen.

Eine *Basislösung* eines linearen Programms ist insbesondere eine solche zulässige Lösung \mathbf{x} , die sich nicht als nicht-triviale Konvexkombination zweier zulässiger Lösungen darstellen lässt: $\mathbf{x} \neq \lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2$, $\lambda > 0$ für alle zulässigen $\mathbf{x}_1, \mathbf{x}_2 \neq \mathbf{x}$.

Untere Schranken aus der LP-Relaxation

Wir werden Optimierungsprobleme als *ganzzahlige* Programme

$$OPT = \min\{\mathbf{c}^T \mathbf{x} \mid A\mathbf{x} \geq \mathbf{b}, \mathbf{x} \in \mathbb{Z}_+^n\}$$

beschreiben. Lassen wir von einer Problembeschreibung einige Restriktionen weg oder weichen sie teilweise auf, dann sprechen wir von einer *Relaxation*. Da sich die Lösungsmenge höchstens vergrößert, wird der optimale Zielfunktionswert höchstens kleiner; wir erhalten so eine untere Schranke LB für OPT . In der Hauptsache wird uns die *LP-Relaxation* eines ganzzahligen Programms interessieren, bei der schlicht $\mathbf{x} \in \mathbb{Z}_+^n$ durch $\mathbf{x} \geq \mathbf{0}$ ersetzt wird. Nach dem schwachen Dualitätssatz ist auch der (insbesondere optimale) Zielfunktionswert des dualen Programms eine untere Schranke für OPT .

Definition 3.4 Sei Π ein Minimierungsproblem formuliert als ganzzahliges Programm mit optimalem Zielfunktionswert OPT und Wert der LP-Relaxation LB . Die Ganzzahligkeitslücke (engl. integrality gap) der LP-Relaxation ist definiert als

$$\sup_{I \in \Pi} \frac{OPT(I)}{LB(I)} .$$

Dies ist ein Maß für die Güte der LP-Relaxation, je näher an 1 desto besser. Eine wichtige Bemerkung: Benutzt man (wie es häufig geschieht) als untere Schranke in der Analyse den Wert einer zulässigen dualen Lösung oder den einer fraktionalen primalen Lösung, so kann man keinen Approximationsfaktor besser als die Ganzzahligkeitslücke erhalten. Das bedeutet auch, dass man sich bei der Formulierung eines Problems als ganzzahliges Programm „Mühe geben“ sollte, eine möglichst gute LP-Relaxation zu erhalten. Natürlich ist auch das eine Sache der Erfahrung. Erneut stellt sich uns eine Frage nach einer unteren Schranke: Wie groß ist die Ganzzahligkeitslücke mindestens (oder auch: höchstens)?

3.2 Runden der fraktionalen Lösung

Möglicherweise *die* erste Idee, aus einer fraktionalen eine ganzzahlige Lösung zu erhalten ist, fraktionale Variablenwerte aufzurunden. Ist der kleinste aufgerundete Wert nach unten durch eine Konstante $1/f$ beschränkt, steigen die Kosten der Lösung höchstens um einen Faktor f . Dies ergibt einen f -Approximationsalgorithmus, wenn die gerundete Lösung zulässig für das ganzzahlige Programm ist.

Für eine Formulierung des gewichteten SET COVER Problems als ganzzahliges Programm führen wir für jede Menge $S \in \mathcal{S}$ eine binäre Variable $x_S \in \{0, 1\}$ ein mit $x_S = 1$ genau dann, wenn S in die Überdeckung aufgenommen wird.

$$\begin{aligned} OPT = \min \quad & \sum_{S \in \mathcal{S}} c(S)x_S \\ & \sum_{S \in \mathcal{S}: e \in S} x_S \geq 1 \quad e \in U \\ & x_S \in \{0, 1\} \quad S \in \mathcal{S} , \end{aligned} \tag{3.2}$$

In der LP-Relaxation dürfen die Variablen kontinuierliche Werte annehmen; die obere Schranke $x_S \leq 1$ müssen wir nicht explizit fordern, weil wir minimieren und $c(S) \geq 0$:

$$\begin{aligned} LB = \min \quad & \sum_{S \in \mathcal{S}} c(S)x_S \\ & \sum_{S \in \mathcal{S}: e \in S} x_S \geq 1 \quad e \in U \\ & x_S \geq 0 \quad S \in \mathcal{S} . \end{aligned} \tag{3.3}$$

Für eine Menge $\mathcal{C} \subseteq \mathcal{S}$ schreiben wir wie üblich $c(\mathcal{C}) = \sum_{S \in \mathcal{C}} c(S)$.

3.2.1 Deterministisch

last edit:
1.11.2006

Definition 3.5 Die Frequenz f eines Mengensystems $\mathcal{S} \subseteq 2^U$ ist die größte Anzahl von Mengen, in denen ein Element $e \in U$ enthalten ist, also $f = \max_{e \in U} |\{S \in \mathcal{S} \mid e \in S\}|$.

Algorithmus 5 LP-Runden für SET COVER

$\mathcal{C} := \emptyset$

Bestimme eine optimale Lösung \mathbf{x} der LP-Relaxation (3.3)

Nimm alle Mengen S in \mathcal{C} auf mit $x_S \geq 1/f$

Satz 3.6 LP-Runden ist ein f -Approximationsalgorithmus für SET COVER.

Beweis. Jedes Element $e \in U$ taucht in höchstens f Mengen auf; eine dieser Mengen muss daher in der LP-Relaxation mindestens mit Wert $1/f$ gewählt werden. Daher ist e durch \mathcal{C} überdeckt, \mathcal{C} ist also eine Überdeckung. Durch Aufrunden wird der Wert einer Variablen x_S , $S \in \mathcal{C}$, höchstens um den Faktor f erhöht, entsprechend sind die Kosten von \mathcal{C} höchstens um einen Faktor f größer als LB ; es gilt $c(\mathcal{C}) \leq f \cdot LB \leq f \cdot OPT$. \square

Übung. Für das VERTEX COVER Problem gilt $f = 2$, daher ist LP-Runden hierfür ein 2-Approximationsalgorithmus.

Beispiel. Die Analyse des LP-Rundens ist scharf: Interpretiere dazu eine Instanz des SET COVER Problems als *Hypergraph* H , dessen Knoten aus dem Mengensystem \mathcal{S} bestehen, und dessen Hyperkanten das Universum U der Elemente darstellen. Eine Menge $S \in \mathcal{S}$ enthält ein Element $e \in U$ genau dann, wenn e und S in H inzidieren. (Dieses Problem ist VERTEX COVER für Hypergraphen.)

Gegeben sei nun folgende Instanz. Seien V_1, \dots, V_f f disjunkte Mengen mit $|V_i| = n$, $i = 1, \dots, f$. Der Hypergraph bekommt als Knotenmenge $V_1 \cup \dots \cup V_f$ und n^f Hyperkanten, die definiert sind als alle f -Tupel, die in der i -ten Komponente ein Element aus V_i haben, $i = 1, \dots, f$. Jeder Knoten kostet 1. Interpretiert als SET COVER Instanz, nimmt man jede Menge mit einem Wert von $1/f$ zu Gesamtkosten von n in die fraktionale Überdeckung auf, weil jede Kante zu genau f Knoten inzident ist. LP-Runden gibt daher *alle* $n \cdot f$ Mengen als approximative Lösung aus, wohingegen die Überdeckung bestehend aus etwa V_1 liefert, dass $OPT = n$.

Die Funktionsweise dieses Beispiels verdeutlichen wir für $f = 3$ und $n = 2$. Bild (a) zeigt die drei Mengen V_1, V_2, V_3 mit je zwei Knoten; die $2^3 = 8$ Hyperkanten sind in (b) schematisiert. Jedes Element (jede Hyperkante) ist in $f = 3$ Mengen (wird von drei Knoten überdeckt). Eine optimale fraktionale Überdeckung enthält *jede* der $nf = 6$ Mengen (jeden Knoten) mit Wert $1/f = 1/3$, daher ist $LB = nf = 6$, wobei bereits V_1 alle Kanten überdeckt: $OPT = |V_1| = 2$. Es ist $OPT/LB = f = 3$.

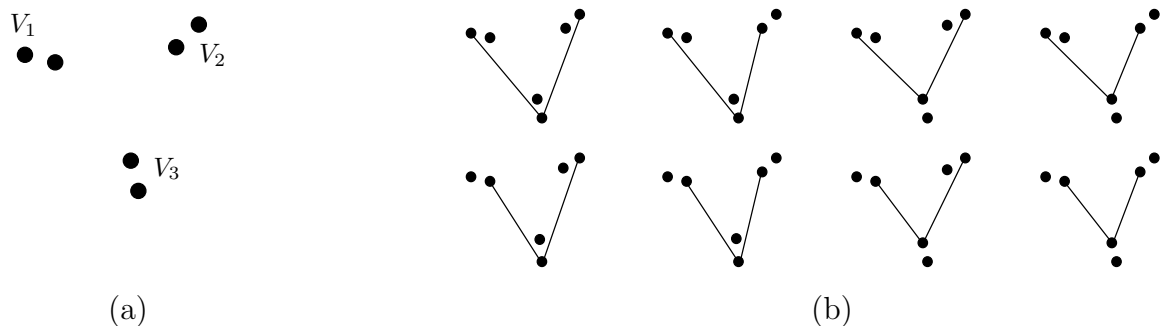


Abbildung 3.1: *Tight example* für LP-Runden für Set Cover

3.2.2 Randomisiert

Es ist naheliegend, den fraktionalen Variablenwert $x_S \in [0, 1]$ als Wahrscheinlichkeit zu interpretieren, mit der die Menge S in die Überdeckung aufgenommen wird.

Definition 3.7 Ein randomisierter α -Approximationsalgorithmus \mathcal{A} für ein Minimierungsproblem Π berechnet zu jeder Instanz $I \in \Pi$ in Zeit polynomial in $|I|$ eine zulässige Lösung $y \in \text{Lsg}(I)$ so, dass $\Pr[v(I, y) \leq \alpha \cdot OPT(I)] \geq 1/2$

Die in dieser Definition aufgeweichte Garantie der Güte ist nur eine Variante, einen randomisierten Approximationsalgorithmus zu definieren. Man könnte genauso die Zulässigkeit

einer Lösung nur mit einer gewissen Wahrscheinlichkeit fordern oder auch die polynomiale Laufzeit nur im Erwartungswert verlangen. Tatsächlich gibt es Beispiele aus der Literatur, bei denen dies geschieht. Auch der nun folgende Algorithmus konstruiert zulässige Lösungen nicht mit Sicherheit.

Randomisiertes Runden ist eine Variante des LP-Rundens, bei der wir *mit hoher Wahrscheinlichkeit* eine Überdeckung erhalten. Sei $x_S =: p_S$, $S \in \mathcal{S}$, eine optimale fraktionale Lösung des SET COVER Problems („fraktionales Cover“) mit Zielfunktionswert LB .

Algorithmus 6 Randomisiertes Runden für SET COVER

$\mathcal{C} = \emptyset$
Wiederhole $O(\log n)$ mal:
for $S \in \mathcal{S}$ **do**
 Nimm die Menge S mit Wahrscheinlichkeit p_S in \mathcal{C} auf
end for

Satz 3.8 *Randomisiertes Runden ist ein randomisierter $O(\log n)$ -Approximationsalgorithmus für SET COVER.*

Beweis. Wir wählen $S \in \mathcal{S}$ mit Wahrscheinlichkeit p_S aus. Sei \mathcal{C}' die entstandene Menge. Deren erwartete Kosten sind

$$\mathbf{E}[c(\mathcal{C}')] = \sum_{S \in \mathcal{S}} \Pr[S \in \mathcal{C}'] \cdot c_S = \sum_{S \in \mathcal{S}} p_S \cdot c_S = LB . \quad (3.4)$$

Wir zeigen jetzt, dass jedes Element $a \in U$ mit konstanter Wahrscheinlichkeit von \mathcal{C}' überdeckt wird. Das Element a sei in k Mengen aus \mathcal{S} enthalten, deren Wahrscheinlichkeiten wir mit p_1, \dots, p_k bezeichnen. Zunächst ist $p_1 + \dots + p_k \geq 1$, denn a ist (fraktional) überdeckt. Eine untere Schranke für die Wahrscheinlichkeit, dass a durch \mathcal{C}' überdeckt wird, erhalten wir, wenn wir $p_i = 1/k$ setzen (kleiner darf die Summe der p_i nicht sein, und wäre ein $p_i > 1/k$, stiege die Wahrscheinlichkeit, dass a überdeckt wird). Daher bekommen wir

$$\Pr[a \text{ ist durch } \mathcal{C}' \text{ überdeckt}] \geq 1 - \left(1 - \frac{1}{k}\right)^k \geq 1 - \frac{1}{e} . \quad (3.5)$$

Um eine Überdeckung *aller* Elemente $a \in U$ zu erhalten, wiederholen wir das Runden $c \log n$ mal und geben die Vereinigung \mathcal{C} aller \mathcal{C}' aus. Die Konstante c sei hier so gewählt, dass

$$\left(\frac{1}{e}\right)^{c \log n} = \frac{1}{n^c} \leq \frac{1}{4n} .$$

Die Gegenwahrscheinlichkeit des Ereignisses in (3.5), $c \log n$ mal wiederholt, ist damit

$$\Pr[a \text{ ist nicht durch } \mathcal{C} \text{ überdeckt}] \leq \left(\frac{1}{e}\right)^{c \log n} \leq \frac{1}{4n} .$$

Über alle $a \in U$ aufsummiert ergibt sich

$$\Pr[\mathcal{C} \text{ ist keine Überdeckung von } U] \leq n \cdot \frac{1}{4n} = \frac{1}{4} . \quad (3.6)$$

Um die Wahrscheinlichkeit abzuschätzen, dass \mathcal{C} „zu teuer“ ist benutzen wir

Lemma 3.9 (Markov-Ungleichung)

Sei X eine nicht-negative Zufallsvariable. Für $t > 0$ gilt $\Pr[X \geq t] \leq \mathbf{E}[X]/t$.

Wegen (3.4) gilt $\mathbf{E}[c(\mathcal{C})] \leq LB \cdot c \log n$. Mit $t = LB \cdot 4c \cdot \log n$ in der Markov-Ungleichung erhalten wir

$$\Pr[c(\mathcal{C}) \geq LB \cdot 4c \cdot \log n] \leq \frac{1}{4} . \tag{3.7}$$

Die Wahrscheinlichkeit, dass die Vereinigung der (nicht erwünschten) Ereignisse in (3.6) und (3.7) eintritt, ist demnach nicht größer als 1/2, oder anders:

$$\Pr[\mathcal{C} \text{ ist eine Überdeckung von } U \text{ mit } c(\mathcal{C}) \leq LB \cdot 4c \cdot \log n] \geq \frac{1}{2} .$$

Trifft eine der beiden (in polynomialer Zeit überprüfbaren!) Bedingungen nicht zu, wiederholen wir den Algorithmus. Die erwartete Anzahl der Durchläufe ist höchstens 2. □

3.3 Dual Fitting

last edit:
7.11.06

„Was nicht passt, wird passend gemacht“, so könnte man das Motto des *Dual fitting* beschreiben. Man konstruiert hierbei eine ganzzahlige primal zulässige Lösung \mathbf{x} und eine duale „Lösung“ $\bar{\mathbf{y}}$, deren Wert $\bar{\mathbf{y}}^T \mathbf{b}$ nicht kleiner ist als $\mathbf{c}^T \mathbf{x}$. Wir sagen auch, die duale Lösung „bezahlt“ für die primale Lösung. Im Allgemeinen ist $\bar{\mathbf{y}}$ nicht dual zulässig (wie sonst könnte sein Wert über dem der primalen Lösung liegen?). Die Technik ist nun, $\bar{\mathbf{y}}$ mittels Division durch einen geeigneten Faktor α in eine dual zulässige (fraktionale) Lösung \mathbf{y} zu überführen, aus der wir den Wert einer unteren Schranke für OPT gewinnen. Man kann auch sagen, wir suchen nach einem $\bar{\mathbf{y}}$, dass jede duale Restriktion höchstens um einen Faktor α verletzt.

Auf dem Zahlenstrahl stellen sich die für uns im Zusammenhang mit linearen Programmen interessanten Zielfunktionswerte und Schranken (beachte: $LB = z_{LP} = z_{DP}$) wie folgt dar:

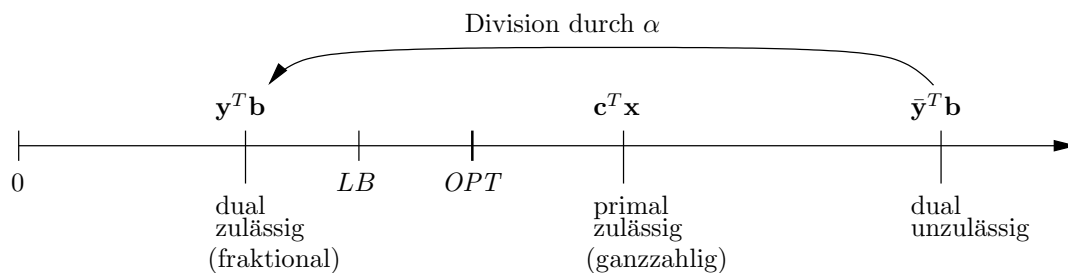


Abbildung 3.2: Vergleich von Schranken und Zielfunktionswerten auf dem Zahlenstrahl

Der Approximationsfaktor α basiert auf der Beobachtung, dass

$$\mathbf{c}^T \mathbf{x} \leq \bar{\mathbf{y}}^T \mathbf{b} = \alpha \cdot \mathbf{y}^T \mathbf{b} \leq \alpha \cdot LB \leq \alpha \cdot OPT .$$

Man würde meinen, dass eine zulässige Lösung des dualen Problems für eine Schranke schlechter brauchbar sei als eine optimale Lösung der primalen LP-Relaxation. Tatsächlich beruhen viele Algorithmen auf einer unteren Schranke aus dem dualen LP, denn sie ist leichter zu bekommen, wenn wir im Approximationsalgorithmus *keine* linearen Programme lösen möchten.

Wir nehmen uns als Anschauung wieder das SET COVER Problem vor und analysieren noch einmal den Greedy-Algorithmus 2 aus dem vorigen Kapitel. Das zu (3.3) duale lineare Programm ist

$$\begin{aligned}
 LB = \max \quad & \sum_{e \in U} y_e \\
 & \sum_{e \in U: e \in S} y_e \leq c(S) \quad S \in \mathcal{S} \\
 & y_e \geq 0 \quad e \in U .
 \end{aligned} \tag{3.8}$$

In manchen Fällen macht es Sinn, das duale Programm auch als *Problem* zu interpretieren. In diesem Fall tun wir dies, um gewisse Sprechweisen einzuführen. Man kann sich hier vorstellen, dass jede Menge $S \in \mathcal{S}$ mit dem Wert der nicht-negativen \mathbf{y} -Variablen „bepackt“ wird, jedoch jeweils nicht mehr („voller“) als $c(S)$.

Definition 3.10 Eine Menge $S \in \mathcal{S}$ mit $\sum_{e \in U: e \in S} y_e = c(S)$ heißt voll. S heißt überpackt, wenn $\sum_{e \in U: e \in S} y_e > c(S)$.

Der Greedy-Algorithmus für SET COVER definiert Preise $price(e)$ für jedes Element $e \in U$. Wir interpretieren die Preise als eine duale Lösung $\bar{\mathbf{y}}$, die für die Überdeckung \mathcal{C} „bezahlt“, denn $c(\mathcal{C}) = \sum_{k=1}^n price(e_k)$ (s. Beweis von Satz 2.3).

Die untere Schranke in der Analyse des Greedy-Algorithmus basiert auf dem folgenden

Lemma 3.11 Der Vektor \mathbf{y} definiert durch

$$y_e = \frac{price(e)}{H_n}, \quad e \in U \tag{3.9}$$

ist eine zulässige Lösung für (3.8).

Beweis. Es muss gezeigt werden, dass keine Menge durch \mathbf{y} überpackt wird. Sei $S \in \mathcal{S}$ eine k -elementige Menge, deren Elemente in der Reihenfolge numeriert seien, in der sie vom Greedy-Algorithmus überdeckt werden: e_1, \dots, e_k .

Am Anfang der Iteration, in der e_i überdeckt wird, enthält S mindestens $k - i + 1$ unüberdeckte Elemente. Zu diesem Zeitpunkt kann S das Element e_i mit durchschnittlichen Kosten $c(S)/(k - i + 1)$ überdecken. Weil e_i von der in dieser Iteration kosteneffektivsten Menge überdeckt wurde, gilt $price(e_i) \leq c(S)/(k - i + 1)$ und damit nach (3.9)

$$y_{e_i} \leq \frac{1}{H_n} \cdot \frac{c(S)}{k - i + 1} .$$

Summation über alle Elemente in S gibt (wegen $H_k \leq H_n$)

$$\sum_{i=1}^k y_{e_i} \leq \frac{c(S)}{H_n} \cdot \left(\frac{1}{k} + \frac{1}{k-1} + \dots + \frac{1}{1} \right) = \frac{H_k}{H_n} \cdot c(S) \leq c(S) . \tag{3.10}$$

□

Satz 3.12 Die Approximationsgarantie des Greedy-Algorithmus 2 für SET COVER ist H_n .

Beweis. Die Kosten der Greedy-Überdeckung sind nach (3.9)

$$c(\mathcal{C}) = \sum_{e \in U} \text{price}(e) = H_n \cdot \sum_{e \in U} y_e \leq H_n \cdot \text{OPT} .$$

Die Ungleichung folgt daraus, dass \mathbf{y} eine dual zulässige Lösung ist, für deren Zielfunktionswert $\sum_{e \in U} y_e \leq LB \leq \text{OPT}$ gilt. \square

Bemerkung. Offenbar würde (3.10) nicht allgemein gelten, wenn wir in (3.9) durch einen Faktor kleiner als H_n teilten. Wir lesen aus der Analyse heraus, dass der Greedy-Algorithmus tatsächlich ein H_k -Approximationsalgorithmus ist, wobei k die Mächtigkeit der größten in \mathcal{S} vorkommenden Menge ist.

Bemerkung. Auf dem Zahlenstrahl in Abbildung 3.2 sind zwei Werte nicht verzeichnet: Der einer dual zulässigen *ganzzahligen* Lösung, und der einer primal zulässigen *fraktionalen* Lösung. Mit letzterer können wir nichts anfangen, weil ihr Wert möglicherweise größer ist als OPT . Eine ganzzahlige dual zulässige Lösung hingegen liegt sogar noch „links vom“ Wert einer fraktionalen dual zulässigen Lösung \mathbf{y} . Viele kombinatorischen Minimierungsprobleme (wie VERTEX COVER) haben ein kombinatorisches duales Maximierungsproblem (wie MATCHING). In diesem Fall erhält man aus kombinatorischen Algorithmen natürlicherweise eine ganzzahlige dual zulässige Lösung. Wir sehen hier, dass die Schranke aus einer fraktionalem dual zulässigen Lösung im Allgemeinen aber *besser* sein kann. Darüberhinaus haben wir eine sehr viel größere Flexibilität, eine solche fraktionale Lösung anzugeben, weil wir die Restriktionen des dualen kombinatorischen Problems „nur fraktional“ einhalten müssen.

Überdeckung eines orthogonalen Polygons mit achsenparallelen Rechtecken

Als potentielle (?) Anwendung des Dual Fitting wollen wir ein orthogonales Polygon P in der Ebene (d.h. dessen Kanten parallel zu den Achsen verlaufen) mit minimaler Anzahl θ achsenparalleler Rechtecke überdecken. Abbildung 3.3 legt (fälschlicherweise) nahe, dass eine solche Überdeckung immer „offensichtlich“ zu finden sei. Vom Standpunkt der Komplexität

dieser Abschnitt ist experimentell, Kommentare sehr willkommen

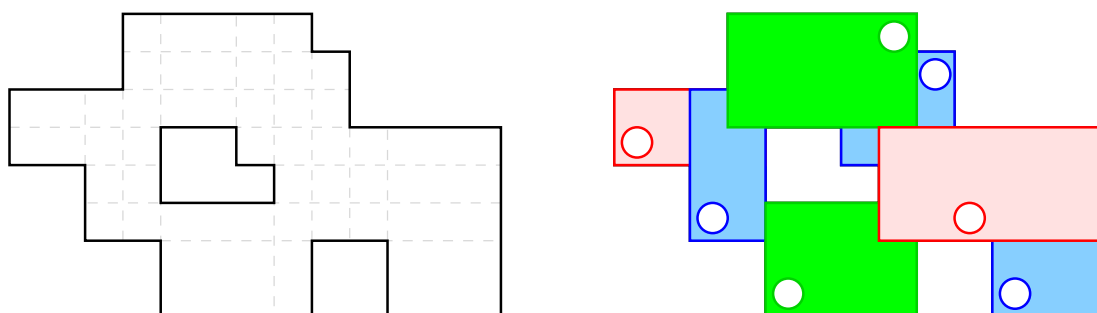


Abbildung 3.3: Überdeckung eines Polygons mit (hier: sieben) Rechtecken. Die Kreise spiegeln eine stabile Menge von Pixeln wider.

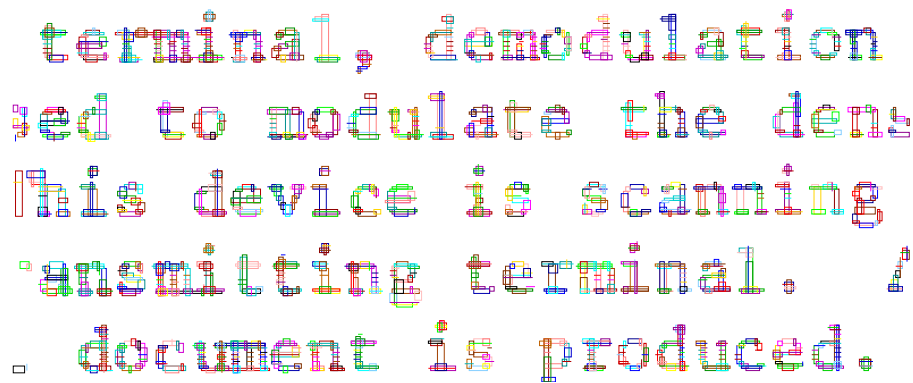


Abbildung 3.4: Ausschnitt aus einem Fax; die (polygonalen) Buchstaben sind mit einer minimalen Anzahl Rechtecken überdeckt

hat das (inzwischen als \mathcal{NP} -schwer bekannte) Problem eine interessante Geschichte; Erdős stellte die bisher nicht gelöste Frage, ob es einen Approximationsalgorithmus mit konstantem Faktor gibt.

Das auf den ersten Blick geometrische Problem ist tatsächlich ein kombinatorisches, wenn wir o.B.d.A. annehmen, dass die Rechtecke inklusionsmaximal (d.h. in Länge und Breite maximal ausgedehnt) sind. Bezeichne R die Menge dieser Rechtecke. Verlängert man alle Kanten von P maximal wie in Abbildung 3.3 gestrichelt angedeutet, so erhält man diskrete Bereiche im Inneren von P , die überdeckt werden müssen, sogenannte *kombinatorische Pixel*. Wir dürfen P daher mit einer Menge von Pixeln identifizieren. Es ist $|P| \in O(n^2)$, wenn n die Anzahl der Kanten bezeichnet. Damit präzisiert sich die Aufgabe dazu, eine minimale Anzahl maximaler Rechtecke zu finden, deren Vereinigung genau P ergibt.

Bisherige Approximationsalgorithmen benutzten als untere Schranke die Kardinalität einer *stabilen Menge*. Dies lehnt sich nicht zufällig an die Terminologie der Graphentheorie an: Man interpretiert die Pixel als Knotenmenge V eines Graphen $G = (V, E)$ mit $(u, v) \in E$ genau dann, wenn u und v in einem gemeinsamen Rechteck liegen („ u und v sehen sich“). Jedes Rechteck repräsentiert dann eine Clique in G , eine Überdeckung mit Rechtecken ist ein *Clique Cover* in G . Zur Bestimmung einer unteren Schranke für θ sucht man in G eine möglichst große stabile Menge, d.h. eine Teilmenge der Knoten/Pixel, die paarweise nicht adjazent sind. Sei α die Kardinalität einer größten stabilen Menge; in Abbildung 3.3 findet man eine solche mit $\alpha = 7 = \theta$. Die eingezeichnete Überdeckung ist somit optimal.

Ein ganzzahliges Programm für dieses spezielle SET COVER Problem ist offensichtlich. Für jedes Rechteck $r \in R$ entscheidet eine Binärvariable x_r darüber, ob wir r in die Überdeckung aufnehmen. Jeder Pixel $p \in P$ muss dabei überdeckt sein.

$$\begin{aligned} \theta &= \min \sum_{r \in R} x_r \\ \text{s. t. } \sum_{r \in R: r \ni p} x_r &\geq 1 && p \in P \\ x_r &\in \{0, 1\} && r \in R \end{aligned}$$

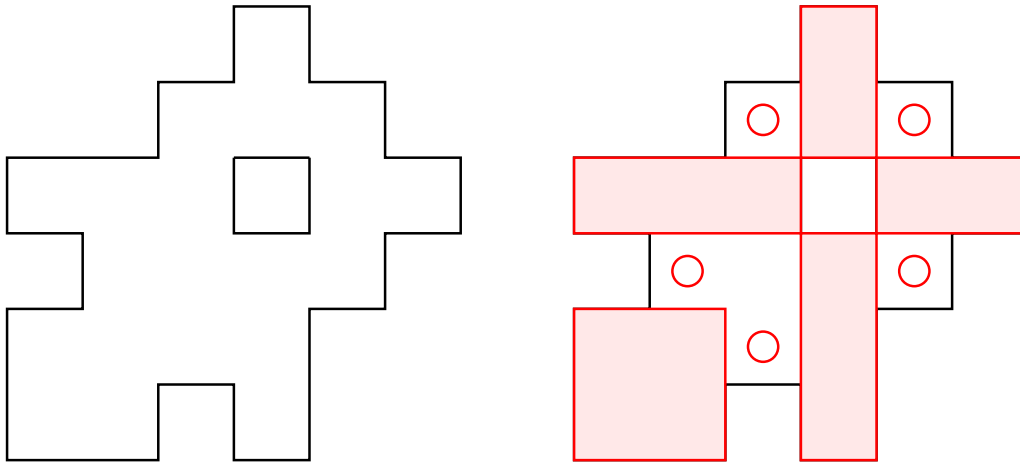


Abbildung 3.5: Ein Polygon mit $7 = \alpha \neq \theta = 8$

Das hierzu duale ganzzahlige Programm ist

$$\begin{aligned} \alpha &= \max \sum_{p \in P} y_p \\ \text{s. t. } \sum_{p \in P: p \in r} y_p &\leq 1 \quad r \in R \\ y_p &\in \{0, 1\} \quad p \in P \end{aligned}$$

Es wurde zunächst vermutet, dass immer $\theta = \alpha$, und das stimmt auch für konvexe Polygone. Im Allgemeinen ist diese Aussage allerdings falsch, wie das Polygon in Abbildung 3.5 zeigt. Die schattierten Rechtecke müssen in jeder Überdeckung vorhanden sein. Die fünf eingezeichneten Pixel induzieren einen Kreis ungerader Länge, einen C_5 . In diesem hat eine maximale stabile Menge Kardinalität 2, eine Überdeckung benötigt aber wenigstens 3 Kanten bzw. Rechtecke. Daher ist $7 = \alpha \neq \theta = 8$.

Eine fraktionale stabile Menge, d.h. eine Lösung der LP-Relaxation des dualen Programms hat einen nicht-kleineren Zielfunktionswert $\bar{\alpha}$ als eine ganzzahlige stabile Menge; daher ist $\bar{\alpha}$ eine potentiell bessere untere Schranke auf $OPT = \theta$. Dual Fitting könnte diese Schranke ausnutzen: Konstruiere eine Überdeckung C und eine Menge S von Pixeln, die für C „bezahlt“. Falls man garantieren kann, dass kein Rechteck mehr als eine konstante Anzahl a der Pixel in S enthält, so liefert $y_e = 1/a$, $e \in S$ und $y_e = 0$, $e \notin S$ eine dual zulässige fraktionale Lösung (und damit über den Dual Fitting Mechanismus einen a -Approximationsalgorithmus).

Trotz einiger verfolgswürdiger Ideen ist es bisher nicht gelungen, ein solches Tripel (C, S, a) zu finden [21].

3.4 Ganz- und Halbzahligkeit

Die optimale Lösung eines linearen Programms kann zufällig ganzzahlig sein. Lineare Relationen $\min\{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \geq \mathbf{1}, \mathbf{x} \geq \mathbf{0}\}$ von SET COVER Problemen haben immer eine ganzzahlige Optimallösung, wenn A *balanciert* ist (d.h. keine Untermatrix ist Inzidenzmatrix eines

ungeraden Kreises). Wir kennen das auch schon von Flussproblemen, bei denen die Knoten-Kanten-Inzidenzmatrix vollständig unimodular ist. Wir wissen aber, dass wir Ganzzahligkeit für \mathcal{NP} -schwere Probleme nicht allgemein erwarten können, denn das hieße $\mathcal{P} = \mathcal{NP}$.

Das „zweitbeste“ wäre ein Beweis, dass eine optimale LP-Lösung \mathbf{x} immer *halbzahlig* ist, also $2\mathbf{x} \in \mathbb{Z}$ gilt. Eine solche Lösung könnte man zu einer ganzzahligen (hoffentlich zulässigen) Lösung aufrunden; die Kosten würden sich höchstens verdoppeln: Eine 2-Approximation. Ein solcher Beweis könnte so aussehen, dass man für jede reguläre Teilmatrix von A zeigt, dass der Betrag ihrer Determinante nur 1 oder 2 sein kann, bzw. allgemeiner nur die Werte $1, \dots, k$ annimmt, um eine k -Approximation zu erzielen. Wir diskutieren eine weitere Idee.

3.4.1 Weighted Vertex Cover

Wir betrachten WEIGHTED VERTEX COVER mit Kostenfunktion $c : V \rightarrow \mathbb{Q}_+$, bei dem eine kostenminimale Überdeckung der Knoten gesucht wird. Es lässt sich als ganzzahliges Programm formulieren:

$$\begin{aligned} OPT = \min \quad & \sum_{v \in V} c(v)x_v \\ & x_u + x_v \geq 1 \quad (u, v) \in E \\ & x_v \in \{0, 1\} \quad v \in V, \end{aligned}$$

dessen LP-Relaxation wie folgt aussieht:

$$\begin{aligned} LB = \min \quad & \sum_{v \in V} c(v)x_v \\ & x_u + x_v \geq 1 \quad (u, v) \in E \\ & x_v \geq 0 \quad v \in V. \end{aligned} \tag{3.11}$$

Satz 3.13 *Jede Basislösung der LP-Relaxation (3.11) ist halbzahlig.*

last edit:
14.11.06

Beweis. Nehmen wir an, eine Basislösung \mathbf{x} sei nicht halbzahlig. Wir zeigen, dass \mathbf{x} dann nicht-triviale Konvexkombination zweier zulässiger Lösungen ist.

Wir betrachten die Knoten $v \in V$, für die $2x_v \notin \mathbb{Z}_+$ und partitionieren diese Menge:

$$V_+ = \{v \mid 1/2 < x_v < 1\}, \quad V_- = \{v \mid 0 < x_v < 1/2\}.$$

Für $\varepsilon > 0$ definieren wir folgende Lösungen:

$$y_v = \begin{cases} x_v + \varepsilon & x_v \in V_+ \\ x_v - \varepsilon & x_v \in V_- \\ x_v & \text{sonst} \end{cases}, \quad z_v = \begin{cases} x_v - \varepsilon & x_v \in V_+ \\ x_v + \varepsilon & x_v \in V_- \\ x_v & \text{sonst} \end{cases}.$$

$V_+ \cup V_- \neq \emptyset$ nach Voraussetzung. Daher ist \mathbf{x} verschieden von \mathbf{y} und \mathbf{z} . Weiterhin ist \mathbf{x} Konvexkombination von \mathbf{y} und \mathbf{z} , denn $\mathbf{x} = \frac{1}{2}(\mathbf{y} + \mathbf{z})$. Es muss nun noch gezeigt werden, dass $\varepsilon > 0$ klein genug gewählt werden kann, dass \mathbf{y} und \mathbf{z} zulässige Lösungen für das LP sind.

Für $\varepsilon < \min_v x_v$ gilt $\mathbf{y}, \mathbf{z} \geq \mathbf{0}$. Für eine Kante $(u, v) \in E$ mit $x_u + x_v > 1$ ist $\{u, v\} \cap V_+ \neq \emptyset$. Daher ist $y_u + y_v > 1$ nach Definition. Wählt man $\varepsilon < \frac{1}{2} \min_{u,v} (x_u + x_v - 1)$, so ist auch

$z_u + z_v > 1$. Bleibt der Fall $x_u + x_v = 1$ für $(u, v) \in E$, der (bis auf Symmetrie) drei Möglichkeiten zulässt: $x_u = x_v = \frac{1}{2}$; $x_u = 0, x_v = 1$; und $u \in V_+, v \in V_-$. Für jede Wahl von $\varepsilon > 0$ gilt jeweils $x_u + x_v = y_u + y_v = z_u + z_v = 1$. Die Behauptung folgt. \square

Weil das Aufrunden aller Variablen $x_v \geq 1/2$ in einer optimalen Lösung der LP-Relaxation zulässig für das ganzzahlige Programm ist, ist LP-Runden ein 2-Approximationsalgorithmus für WEIGHTED VERTEX COVER.

Beispiel. Betrachten wir wieder K_n , mit $c(v) = 1, v \in V$. Wie im ungewichteten Fall ist $OPT = n - 1$. Die LP-Relaxation hat als zulässige Lösung $x_v = 1/2, v \in V$, und daher $LB \leq n/2$. Die Ganzzahligkeitslücke ist asymptotisch mindestens 2.

Bemerkung. Man muss nicht notwendigerweise ein lineares Programm für diesen Approximationsalgorithmus lösen. Es gibt einen kombinatorischen Algorithmus zum Auffinden einer halbzahligen fraktionalen Knotenüberdeckung. Wir lernen im nächsten Kapitel eine kombinatorische Alternative kennen, daher belassen wir es an dieser Stelle dabei.

Kapitel 4

Primal-Duales Schema

last edit:
14.11.06

Aus dem primal-dualen Verfahren, ursprünglich zum Lösen linearer Programme entwickelt, sind für polynomial lösbare kombinatorische Optimierungsprobleme schöne Algorithmen hervorgegangen. Ausgehend von einer dual zulässigen Lösung versucht man, eine primale Lösung zu konstruieren, die den Satz vom komplementären Schlupf erfüllt. Im Erfolgsfall hat man einen Beweis der Optimalität, ansonsten wird eine duale Lösung mit einem größeren Zielfunktionswert gesucht und iteriert. Eine Relaxierung dieser Ideen hat sich als *die* generische Methode überhaupt zur Entwicklung von Approximationsalgorithmen erwiesen und wurde formal zum ersten Mal von Bar-Yehuda und Even [8] eingesetzt.

4.1 Komplementärer Schlupf relaxiert

Satz 4.1 Seien $\alpha, \beta \geq 1$. Seien $\mathbf{x} \in \mathbb{R}^n$ und $\mathbf{y} \in \mathbb{R}^m$ primal bzw. dual zulässige Lösungen, die die folgenden relaxierten Bedingungen vom Komplementären Schlupf erfüllen:

Primale Bedingungen: Für $j = 1, \dots, n$ gilt: $x_j = 0$ oder $\frac{c_j}{\alpha} \leq \sum_{i=1}^m a_{ij}y_i \leq c_j$.

Duale Bedingungen: Für $i = 1, \dots, m$ gilt: $y_i = 0$ oder $b_i \leq \sum_{j=1}^n a_{ij}x_j \leq \beta \cdot b_i$.

Dann gilt $\mathbf{c}^T \mathbf{x} \leq \alpha \cdot \beta \cdot \mathbf{y}^T \mathbf{b}$.

Beweis. Es gilt $\mathbf{c}^T \mathbf{x} \leq \alpha \cdot \mathbf{y}^T \mathbf{A} \mathbf{x} \leq \alpha \cdot \beta \cdot \mathbf{y}^T \mathbf{b}$, wobei die erste Ungleichung aus den relaxierten primalen, die zweite Ungleichung aus den relaxierten dualen Bedingungen stammt. \square

Wir werden in dieser Vorlesung nur die dualen Bedingungen relaxieren, d.h. wir benutzen diesen Satz mit $\alpha = 1$. Haben wir eine ganzzahlige primale und eine zulässige duale Lösung konstruiert, die zusammen mit einem geeigneten $\beta \geq 1$ die Bedingungen des Satzes erfüllen, so benutzen wir den dualen Zielfunktionswert als untere Schranke für OPT und erhalten einen β -Approximationsalgorithmus.

Bemerkung. Man beachte, dass $\mathbf{c}^T \mathbf{x} \leq \alpha \cdot \mathbf{y}^T \mathbf{A} \mathbf{x}$ auch gelten kann, wenn einige relaxierte primale Bedingungen *nicht* erfüllt sind, da hier ein mit \mathbf{x} gewichteter Durchschnitt betrachtet

wird (und wir ggf. Informationen über \mathbf{x} haben). Entsprechendes gilt für die dualen Bedingungen. Wir lernen dies noch in aller Form kennen.

4.2 Grundversion Primal-duales Schema

Das HITTING SET Problem erhält als Eingabe eine Grundmenge $E = \{e_1, \dots, e_n\}$ an Elementen mit zugehörigen Kosten $c_e \geq 0$, $e \in E$ sowie Teilmengen $T_1, \dots, T_p \subseteq E$. Gesucht ist eine kostenminimale Teilmenge $A \subseteq E$ der Elemente, die jedes T_i „trifft“, d.h. $T_i \cap A \neq \emptyset$ für jedes $1 \leq i \leq p$. In dieser Allgemeinheit umfasst es leichte wie schwere Probleme, u.a.

- KÜRZESTER WEG in ungerichteten Graphen
- PERFEKTES MATCHING
- SPANNENDER BAUM
- STEINERBAUM und Verallgemeinerungen
- VERTEX COVER
- SURVIVABLE NETWORK DESIGN

Vertauschen wir die Rollen von „Element“ und „Menge“, so sieht man, dass HITTING SET sogar äquivalent zu SET COVER ist. Die natürliche Formulierung von HITTING SET als ganzzahliges Programm ist:

$$\begin{aligned}
 OPT = \min \quad & \sum_{e \in E} c_e x_e \\
 & \sum_{e \in T_i} x_e \geq 1 \quad i = 1, \dots, p \\
 & x_e \in \{0, 1\} \quad e \in E,
 \end{aligned} \tag{4.1}$$

wobei $x_e = 1 \iff e \in A$. Für die LP-Relaxation fordern wir in (4.1) wieder nur $\mathbf{x} \geq \mathbf{0}$. Das zur Relaxation duale lineare Programm lautet:

$$\begin{aligned}
 \max \quad & \sum_{i=1}^p y_i \\
 & \sum_{i: e \in T_i} y_i \leq c_e \quad e \in E \\
 & y_i \geq 0 \quad i = 1, \dots, p.
 \end{aligned} \tag{4.2}$$

Unser Plan ist es, gleichzeitig eine ganzzahlige zulässige Lösung zu (4.1) und eine dual zulässige (fraktionale) Lösung zu (4.2) zu konstruieren. Typischerweise geht man von den primal unzulässigen, aber dual zulässigen Lösungen $\mathbf{x} = \mathbf{0}$ und $\mathbf{y} = \mathbf{0}$ aus. Für eine ganzzahlige primale Lösung lassen sich die Bedingungen vom Komplementären Schlupf interpretieren als

$$e \in A \implies \sum_{i: e \in T_i} y_i = c_e \tag{4.3}$$

und

$$y_i > 0 \implies |A \cap T_i| = 1. \tag{4.4}$$

Wir lesen in der ersten Bedingung (4.3) ein notwendiges Kriterium für Elemente $e \in E$, die wir in eine primale Lösung $A \subseteq E$ aufnehmen wollen: Höchstens diejenigen, deren duale Restriktion mit Gleichheit erfüllt ist. Erhalten wir durch Setzen von $x_e = 1$ für alle $e \in E$ mit $\sum_{i:e \in T_i} y_i = c_e$ keine primal zulässige Lösung A , d.h. gibt es ein T_k mit $A \cap T_k = \emptyset$, so gibt es kein zulässiges ganzzahliges \mathbf{x} , das (4.3) und (4.4) erfüllt. Wir nennen T_k *verletzt*.

Wir erhalten aus einer verletzten Menge T_k unmittelbar einen Hinweis darauf, wie wir unsere duale Lösung verändern müssen: Für alle $e \in T_k$ gilt $\sum_{i:e \in T_i} y_i < c_e$ (sonst wäre $e \in A$). Weil dies genau die Restriktionen sind, die y_k enthalten können wir y_k auf

$$\min_{e \in T_k} \{c_e - \sum_{i:e \in T_i} y_i\} > 0 \quad (4.5)$$

erhöhen, und \mathbf{y} bleibt dual zulässig. Danach sind die Restriktionen im dualen LP (4.2) für mindestens ein Element $e \in E \setminus A$ mit Gleichheit erfüllt („voll“), und wir können A um e erweitern und iterieren. Falls Gleichheit für eine Menge E' von Elementen gilt, so erlauben wir auch, dass A um E' erweitert wird (man überlegt sich leicht, dass dies ist kein prinzipieller Unterschied ist). Wir fassen unsere Überlegungen als Algorithmus zusammen, bei dem ausdrücklich darauf hingewiesen sei, dass hier an keiner Stelle ein lineares Programm gelöst wird.

Algorithmus 7 Primal-Duales Schema für HITTING SET (Basisversion)

```

y = 0
A =  $\emptyset$ 
while A ist unzulässig do
  Finde verletzte Menge  $T_k$ , d.h.  $A \cap T_k = \emptyset$ 
  Erhöhe  $y_k$ , bis für ein  $e \in T_k$  gilt:  $\sum_{i:e \in T_i} y_i = c_e$ 
  A =  $A \cup \{e\}$ 
end while
Gib A (und y) aus.

```

Laufzeit und Approximationsgüte

last edit:
15.11.06

Weil $A \subseteq E$, terminiert der Algorithmus nach höchstens $|E|$ Iterationen. Ebenso wird in jeder Iteration nur ein y_k erhöht, der Algorithmus muss demnach nur über höchstens $|E|$ positive Einträge aus einem möglicherweise exponentiell langen Vektor \mathbf{y} Buch führen. Wir benötigen schließlich ein *Orakel*, das zu gegebenem $A \subseteq E$ entscheidet, ob A eine zulässige Lösung des HITTING SET Problems ist, und falls nicht, ein verletztes T_k , d.h. $A \cap T_k = \emptyset$ zurückgibt. Man sagt, das Primal-duale Schema für HITTING SET ist *orakel-polynomial*.

Nach Konstruktion gilt für die Kosten von A

$$c(A) = \sum_{e \in A} c_e = \sum_{e \in A} \sum_{i:e \in T_i} y_i$$

weil wir in jeder Iteration (4.3) berücksichtigen. Vertauscht man die Summation, erhält man

$$c(A) = \sum_{i=1}^p |A \cap T_i| y_i .$$

Können wir nun garantieren, dass

$$y_i > 0 \implies |A \cap T_i| \leq \beta \quad \text{für alle } i = 1, \dots, p \quad (4.6)$$

(das sind die relaxierten dualen Bedingungen vom Komplementären Schlupf!), so erhalten wir wegen der dualen Zulässigkeit von \mathbf{y} und somit $\sum_{i=1}^p y_i \leq OPT$ eine β -Approximation:

$$c(A) \leq \beta \cdot \sum_{i=1}^p y_i \leq \beta \cdot OPT .$$

Offensichtlich können wir immer $\beta = \max_{i=1, \dots, p} |T_i|$ wählen.

Beispiel. Beim VERTEX COVER Problem müssen Mengen $T_i = \{u, v\}$ (Kanten (u, v)) von Elementen $v \in V$ (Knoten) knotengewichtsminimal „getroffen“ werden. In diesem Fall gilt $|T_i| = 2$ für $i = 1, \dots, p$. Insbesondere gibt es nur polynomial viele Mengen, wir finden also in Polynomialzeit ein verletztes T_k ; es folgt

Satz 4.2 *Das Primal-duale Schema ist ein 2-Approximationsalgorithmus für GEWICHTETES VERTEX COVER.*

Goemans und Williamson [16] geben in ihrem lesenswerten Überblicksartikel Regeln für die Entwicklung von Approximationsalgorithmen basierend auf dem Primal-dualen Schema an. Deren Beachtung ist nicht zwingend, in Beweisen kann man sie aber ausnutzen; sie leiten sich überzeugend aus *exakten* Algorithmen für klassische kombinatorische Optimierungsprobleme in \mathcal{P} her.

Erste Regel: Wähle eine bezüglich Inklusion minimal verletzte Menge.

Beispiel. Betrachten wir das Primal-duale Schema für das Problem UNGERICHTETER s - t -KÜRZESTER WEG mit nicht-negativen Kantengewichten c_e , $e \in E$. Jeder Weg von s nach t trifft jeden s - t -Schnitt. Man assoziiert mit jeder Knotenteilmenge $S \subseteq V$ mit $s \in S$ und $t \notin S$ den Schnitt $T_S = \delta(S)$. Anfangs setzen wir $\mathbf{y} = \mathbf{0}$ und $A = \emptyset$ und eine (bzgl. Knoteninklusion) minimal verletzte Menge ist $\delta(s)$ (die Sprechweise etwas ausdehnend sagen wir auch, dass $S = \{s\}$ verletzt sei). Erhöhen wir y_s , so wird eine kürzeste Kante $(s, i) \in E$ in A aufgenommen. Wenn im Weiteren eine Kante $(i, j) \in \delta(S)$ für eine minimal verletzte Menge S in A aufgenommen wird, ist eine nächste minimal verletzte Menge $S \cup \{j\}$. Eine genauere Analyse [16] zeigt, dass hier der Dijkstra-Algorithmus initiiert wird mit der einzigen Ausnahme, dass A eine Obermenge eines kürzesten Weges ist.

Rückwärtlöschen und Minimale Erweiterungen

Wir hatten schon früher den Eindruck, eine Lösung durch Entfernen von Teilen der Lösung noch im Nachhinein verbessern zu können, nur haben wir das bisher nicht formalisiert. Tatsächlich ist beim Primal-dualen Schema jedes Element von A im Moment des Einfügens nötig für die Zulässigkeit; nach Beendigung des Algorithmus können aber Elemente redundant sein, die die Kosten der Lösung erhöhen.

Zweite Regel: Entferne redundante Elemente aus A in der umgekehrten Reihenfolge ihres Einfügens, sogenanntes *Rückwärtslöschen*.

Algorithmus 8 Primal-Duales Schema für HITTING SET mit Rückwärtslöschen

$\mathbf{y} = \mathbf{0}$
 $A = \emptyset$
 $\ell = 0$
while es existiert ein T_k mit $A \cap T_k = \emptyset$ **do**
 $\ell = \ell + 1$
 Erhöhe y_k , bis für ein $e_\ell \in T_k$ gilt: $\sum_{i: e_\ell \in T_i} y_i = c_{e_\ell}$
 $A = A \cup \{e_\ell\}$
end while
for $j = \ell$ **downto** 1 **do**
 Falls $A \setminus \{e_j\}$ zulässig, setze $A = A \setminus \{e_j\}$.
end for
Gib $A_f = A$ (und \mathbf{y}) aus.

Für die Analyse wollen wir nun β in (4.6) für diese Variante des Algorithmus beschränken. Bezeichne A_f die rückwärtsgelöschte Lösung. Sei T_i eine verletzte Menge in der Iteration, in der Element e_j in A aufgenommen wurde. Nehmen wir an, dass e_j nicht beim Rückwärtslöschen entfernt wird. Nach Konstruktion ist dann $\{e_1, \dots, e_{j-1}\} \cap T_i = \emptyset$.

Im Moment des Überprüfens von e_j wurde noch kein Element aus $\{e_1, \dots, e_{j-1}\}$ gelöscht. Wir bezeichnen mit $B = A_f \cup \{e_1, \dots, e_{j-1}\}$ die Menge aller zu diesem Zeitpunkt noch nicht gelöschten Elemente. B ist eine sogenannte *minimale Erweiterung von $\{e_1, \dots, e_{j-1}\}$* , d.h. B ist eine zulässige Obermenge, aber für alle $e \in B \setminus \{e_1, \dots, e_{j-1}\}$ ist $B \setminus \{e\}$ unzulässig (nach Konstruktion des Rückwärtslöschens). Weil $B \supseteq A_f$ gilt $|A_f \cap T_i| \leq |B \cap T_i|$.

Intuitiv ist B (mit dem Wissen des weiteren Ablaufs des Algorithmus) eine beste Erweiterung der zu diesem Zeitpunkt konstruierten Teillösung. Der folgende Satz formalisiert diese Intuition für alle möglichen Teillösungen und alle möglichen minimalen Erweiterungen.

Satz 4.3 Für eine im Verlauf des Primal-dualen Schemas mit Rückwärtslöschen unzulässige Zwischenlösung A bezeichnen wir mit $T(A)$ die daraufhin gefundene verletzte Menge. Für

$$\beta = \max_{\substack{\text{unzulässiges} \\ A \subseteq E}} \max_{\substack{\text{minimale} \\ \text{Erweiterung} \\ B \text{ von } A}} |B \cap T(A)| \quad (4.7)$$

gilt

$$c(A_f) = \sum_{i=1}^p |A_f \cap T_i| y_i \leq \beta \cdot \sum_{i=1}^p y_i \leq \beta \cdot \text{OPT} .$$

Beweis. Folgt aus $|A_f \cap T_i| \leq \max_B |B \cap T_i| \leq \beta$. □

Man beachte, dass wir nur deswegen minimale Erweiterungen als Argument heranziehen konnten, weil wir die Reihenfolge des Löschens redundanter Elemente nicht willkürlich gewählt haben (sondern die zweite Regel beachtet). Obwohl es aufwändig klingt, eine so globale Aussage wie (4.7) zu garantieren, kann es im Einzelfall „ganz einfach“ sein.

Beispiel. Sei beim s - t -KÜRZESTER WEG Problem eine noch unzulässige Zwischenlösung A gegeben; sei $T(A)$ ein bzgl. Kanteninklusion minimaler verletztter Schnitt. Eine minimale Erweiterung B von A muss einen s - t -Weg enthalten, der aber durch Entfernen einer Kante aus $B \setminus A$ unterbrochen wird. Daher kann B auch nur eine Kante mit $T(A)$ gemeinsam haben. Weil wir weiter keine Annahmen an A , $T(A)$ und B gemacht haben, haben wir (4.7) für diesen Fall bestimmt. Es folgt

Satz 4.4 *Das Primal-duale Schema mit Rückwärtzlöschern ist ein 1-Approximationsalgorithmus (optimal) für s - t -KÜRZESTER WEG.*

4.3 Minimum Multicut auf Bäumen

Das Problem MINIMUM MULTICUT verallgemeinert s - t -Schnitte. Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit Kantengewichten $c : E \rightarrow \mathbb{Q}_+$, sowie Paare $(s_1, t_1), \dots, (s_p, t_p)$ mit $s_i \neq t_i$. Gesucht ist eine kostenminimale Kantenmenge A , so dass in $(V, E \setminus A)$ jeweils s_i und t_i , $i = 1, \dots, p$ in verschiedenen Zusammenhangskomponenten sind. Dieses Problem ist \mathcal{NP} -schwer für $p \geq 3$. Im Allgemeinen ist das Problem $O(\log p)$ -approximierbar.

last edit:
21.11.06

Wir betrachten hier den Spezialfall, dass G ein Baum ist. Dann gibt es jeweils zwischen s_i und t_i einen eindeutigen Weg, aus dem eine Kante entfernt (und in A aufgenommen) werden muss. Wie zuvor führen wir eine binäre Variable x_e , $e \in E$, ein mit $x_e = 1 \iff e \in A$. Bezeichne T_i die Menge der Kanten entlang des eindeutigen Weges von s_i nach t_i . Dann ist die Formulierung des MINIMUM MULTICUT Problems als ganzzahliges Programm exakt (4.1).

Eine Lösung der zugehörigen LP-Relaxation nennen wir *fraktionales Multicut*: Entlang jedes Weges T_i ist die Summe der fraktionell gewählten Kanten mindestens 1.

Beispiel. Wie das klassische Beispiel in folgender Abbildung 4.1 zeigt, kann eine fraktionale optimale Lösung günstiger sein als eine ganzzahlig optimale: Jede Kante mit Wert $1/2$ gewählt ergibt einen zulässigen fraktionales Multicut, während für eine ganzzahlige Lösung mindestens zwei Kanten entfernt werden müssen. Die Ganzzahligkeitslücke ist also wenigstens $4/3$.

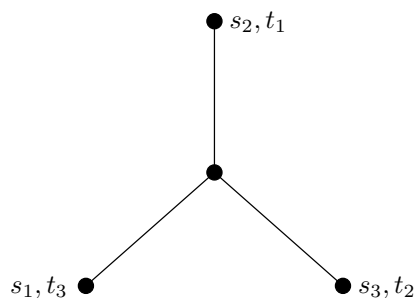


Abbildung 4.1: Klassisches Beispiel zum fraktionales Multicut/Multicommodity Flow

Wie beim s - t -SCHNITT Problem, interpretieren wir auch das duale lineare Programm (4.2) der LP-Relaxation als Flussproblem: Als MULTICOMMODITY FLOW Problem mit p Gütern (eins für jedes Paar (s_i, t_i)) in G . Die Dualvariable y_i gibt an, wieviel (fraktionales) von Gut i entlang des Weges von s_i nach t_i geschickt wird, wobei der Fluss insgesamt maximiert werden

soll, ohne dass die Kantenkapazitäten c_e , $e \in E$, überschritten werden. Dieses duale Programm hat folglich auch eine sinnvolle Interpretation als *ganzzahliges* Programm, bei dem ein GANZZAHLIGER MULTICOMMODITY FLOW gesucht wird: Ein praktisch sehr wichtiges \mathcal{NP} -schweres Problem mit vielen Anwendungen zum Beispiel in Fahrzeugumlauf- und Personalplanung oder in der Telekommunikation. Wählen wir auf jeder Kante des Graphen in Abbildung 4.1 eine Gesamtkapazität von 1, so kann zwischen den Paaren (s_i, t_i) , $i = 1, 2, 3$, jeweils eine halbe Einheit des Gutes i geschickt werden, während ein ganzzahliger Multicommodity Flow höchstens eine Einheit nur eines Gutes schicken darf.

Wir wollen nun das Primal-duale Schema mit Rückwärtlöschen auf das MINIMUM MULTICUT Problem auf Bäumen anwenden. Hierzu müssen wir erklären, wie eine verletzte Menge gefunden wird, also ein s_i - t_i -Weg T_i , von dessen Kanten keine in A aufgenommen ist. Wir wählen einen beliebigen Knoten r als Wurzel und definieren die *Tiefe* eines Knotens v als Anzahl der Kanten des Weges von r nach v . Der *erste gemeinsame Vorgänger* der Knoten u und v ist der Knoten $\wedge(u, v)$ mit geringster Tiefe auf dem Weg von u nach v . Wir wählen immer die verletzte Menge T_k mit *tiefstem* (d.h. am weitesten von der Wurzel entferntesten) $\wedge(s_k, t_k)$. Damit können wir den Algorithmus angeben, der auf Garg, Vazirani und Yannakakis [13] zurückgeht.

Algorithmus 9 Primal-Duales Schema für MINIMUM MULTICUT auf Bäumen

```

y = 0
A =  $\emptyset$ 
while A ist kein Multicut do
    schicke maximalen Fluss von  $s_k$  nach  $t_k$ ,
    wobei  $T_k$  die verletzte Menge mit tiefstem  $\wedge(s_k, t_k)$  ist
    (dies entspricht: Erhöhe  $y_k$ , bis für ein  $e \in T_k$  gilt:  $\sum_{i:e \in T_i} y_i = c_e$ )
    Nimm in A alle in dieser Iteration gesättigten Kanten  $e$  auf
end while
Sei  $e_1, \dots, e_\ell$  die Liste der Kanten in A, geordnet in der Reihenfolge des Einfügens
for  $j = \ell$  downto 1 do
    Falls  $A \setminus \{e_j\}$  ein Multicut ist, setze  $A = A \setminus \{e_j\}$ .
end for
Gib den Multicut  $A_f = A$  (und den Multicommodity Flow y) aus.

```

Satz 4.5 *Das Primal-duale Schema für MINIMUM MULTICUT auf Bäumen ist ein 2-Approximationsalgorithmus.*

Beweis. Wir ziehen Satz 4.3 heran und zeigen, dass $\beta \leq 2$ in (4.7) gilt; s. Abbildung 4.2. Sei A eine unzulässige Lösung und T_k die vom Algorithmus gewählte verletzte Menge, d.h. ein Weg von s_k nach t_k mit tiefstem erstem gemeinsamen Vorgänger $\wedge(s_k, t_k)$. Sei B eine minimale Erweiterung von A (weil wir rückwärts löschen, dürfen wir davon ausgehen, dass die am Ende des Algorithmus konstruierte Lösung eine minimale Erweiterung von A sein wird).

Bezeichne T den Weg von s_k nach $\wedge(s_k, t_k)$. Wir zeigen, dass $|B \cap T| \leq 1$ (und symmetrisch für den Weg von $\wedge(s_k, t_k)$ nach t_k), so dass insgesamt $|B \cap T_k| \leq 2$ und die Behauptung des Satzes folgt. Nehmen wir das Gegenteil an, also $|B \cap T| \geq 2$. Dann ist aber B keine minimale

Erweiterung von A , weil alle Kanten aus $B \cap T$ aus B entfernt werden können außer derjenigen $e' \in B \cap T$ am nächsten an $\wedge(s_k, t_k)$ (d.h. die Kante in $B \cap T$ mit geringster Tiefe). Dies sieht man folgendermaßen ein. Für jeden verletzten Weg T_j mit $T_j \cap T \neq \emptyset$ ist $T_j \cap T$ ein Weg von einem Knoten in T_j nach $\wedge(s_k, t_k)$; andernfalls (s. Abbildung 4.2) wäre $\wedge(s_j, t_j)$ tiefer als $\wedge(s_k, t_k)$ und T_k nicht als verletzte Menge ausgewählt worden, sondern T_j . Wenn also T_j irgendeine Kante aus $B \cap T$ enthält, dann ist auch $e' \in T_j$ (also ist e' bereits ausreichend für eine minimale Erweiterung): Alle Kanten in $(B \cap T) \setminus \{e'\}$ können entfernt werden. Widerspruch zur Minimalität von B . \square

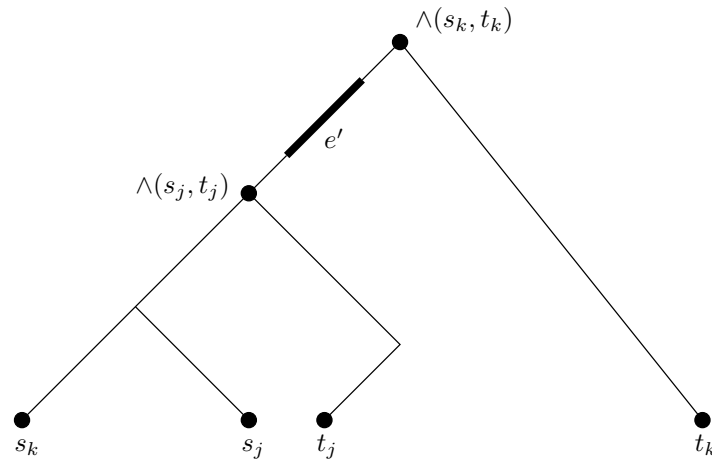


Abbildung 4.2: Zum Widerspruch im Beweis von Satz 4.5

Wir beobachten, dass die duale Lösung \mathbf{y} automatisch ganzzahlig ist, wenn die Kapazitäten $c_e, e \in E$ ganzzahlig sind. In diesem Fall konstruiert der Algorithmus gleichzeitig einen zulässigen ganzzahligen Multicommodity Flow, dessen Wert höchstens um einen Faktor $\frac{1}{2}$ unter dem Wert des ausgegebenen Multicuts liegt. Wir folgern

Satz 4.6 *Das Primal-duale Schema für MINIMUM MULTICUT auf Bäumen ist ein $\frac{1}{2}$ -Approximationsalgorithmus für MAXIMALER GANZZAHLIGER MULTICOMMODITY FLOW auf Bäumen.*

Die Formulierung des GANZZAHLIGEN MULTICOMMODITY FLOW Problems als

$$\begin{aligned} \max \quad & \sum_{i=1}^p y_i \\ & \sum_{i:e \in T_i} y_i \leq c_e \quad e \in E \\ & y_i \in \{0, 1\} \quad i = 1, \dots, p. \end{aligned} \tag{4.8}$$

mit linearer Relaxierung (4.2) führt allerdings für allgemeine Graphen auf keine konstant beschränkte Approximationsgüte. Dies gilt sogar für planare Graphen.

Beispiel. Der Graph in Abbildung 4.3 mit n Paaren $(s_1, t_1), \dots, (s_n, t_n)$ habe Einheitskapazitäten auf allen Kanten. Je zwei Wege zwischen (s_i, t_i) und (s_j, t_j) haben eine Kante

gemeinsam, so dass das Verschicken einer Einheit zwischen einem beliebigen Paar alle anderen Paare blockiert: $OPT = 1$. Eine fraktionale Lösung hingegen kann von s_i nach $t_i, i = 1, \dots, n$ je eine halbe Einheit Fluss schicken, insgesamt also $n/2$ Einheiten. Die Ganzzahligkeitslücke von (4.8) ist mindestens $n/2$, also linear in $|V|$.

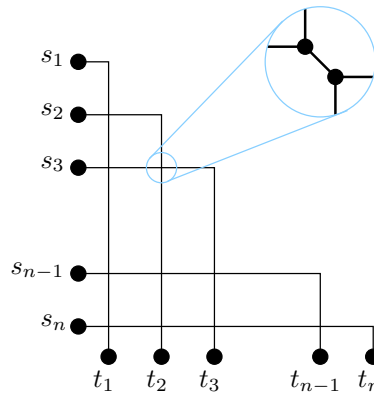


Abbildung 4.3: Zur Ganzzahligkeitslücke von (4.8) in planaren Graphen. Das Detail zeigt, wie alle „Kreuzungen“ planar realisiert werden.

Offenes Problem 4.7 Welche nicht-triviale Approximationsgüte ist möglich für MAXIMALER GANZZAHLIGER MULTICOMMODITY FLOW auf Graphen allgemeiner als Bäume?

Bemerkung. Im Buch von Vazirani [36] werden die Probleme dieses Kapitels (und viele weitere) nicht notwendigerweise im größeren HITTING SET-Kontext eingeführt, so wie das hier in Anlehnung an [16] geschehen ist. Seine Darstellung hat den Vorteil, dass zunächst für jedes Problem die relaxierten Bedingungen vom komplementären Schlupf explizit hingeschrieben und interpretiert werden, und so die grobe Idee für einen Algorithmus (und was für die Güte zu beweisen ist) gegeben wird. Im Fall MINIMUM MULTICUT liest sich das so:

Primale Bedingungen: Für jede Kante $e \in E$ gilt: $x_e \neq 0 \implies \sum_{i:e \in T_i} y_i = c_j$,

d.h. jede Kante im Multicut muss vom Fluss gesättigt sein.

Duale Bedingungen: Für $i = 1, \dots, p$ gilt: $y_i \neq 0 \implies \sum_{e \in T_i} x_j \leq 2$,

d.h. höchstens zwei Kanten eines flussführenden Weges dürfen in den Multicut aufgenommen werden.

4.4 Primal-Duales Schema mit Synchronisierung

Beim Problem MINIMAL SPANNENDER BAUM ist jeder Schnitt eine von den gewählten Kanten der Lösung „zu treffende Menge.“ Jede Zusammenhangskomponente (eingangs also jeder Knoten) induziert eine minimal verletzte Menge, deren Dualvariable erhöht werden kann. Beim Kruskal-Algorithmus wird jeweils die kostengünstigste Kante zwischen je zwei Komponenten, also zwischen zwei verschiedenen minimal verletzten Mengen gewählt. Das legt nahe:

Dritte Regel: Erhöhe die Dualvariablen *aller* minimal verletzten Mengen gleichzeitig mit derselben Geschwindigkeit, genannt *Synchronisierung*.

Algorithmus 10 Primal-Duales Schema für HITTING SET mit Synchronisierung

```

y = 0
A = ∅
ℓ = 0
while A ist nicht zulässig do
    ℓ = ℓ + 1
     $\mathcal{V}(A)$  = Menge aller minimal verletzten Mengen
    Erhöhe  $y_k$  gleichzeitig und gleichmäßig für alle  $T_k \in \mathcal{V}(A)$  bis  $\exists e_\ell \notin A: \sum_{i: e_\ell \in T_i} y_i = c_{e_\ell}$ 
    A = A ∪ { $e_\ell$ }
end while
for  $j = \ell$  downto 1 do
    Falls  $A \setminus \{e_j\}$  zulässig, setze  $A = A \setminus \{e_j\}$ .
end for
Gib  $A_f = A$  (und y) aus.

```

Wir haben bisher die Kosten $c(A_f) = \sum_{i=1}^p |A_f \cap T_i| y_i$ mit dem Wert $\sum_{i=1}^p y_i$ der dualen Lösung verglichen, um eine Approximationsgarantie zu erhalten. Anstatt dies summandenweise zu tun, nutzen wir in der Argumentation, dass mehrere y_i zur selben Zeit um denselben Betrag erhöht werden und betrachten eine Durchschnittsbildung.

Satz 4.8 *Das Primal-duale Schema mit Synchronisierung ist ein γ -Approximationsalgorithmus für HITTING SET, falls für jede unzulässige Teillösung A und jede minimale Erweiterung B von A gilt, dass*

$$\sum_{T_i \in \mathcal{V}(A)} |B \cap T_i| \leq \gamma |\mathcal{V}(A)|, \quad (4.9)$$

wobei $\mathcal{V}(A)$ die Menge aller minimal verletzten Mengen zu A bezeichne.

Beweis. Bezeichne \mathcal{V}_j die Menge aller in der j -ten Iteration minimal verletzen Mengen und sei ϵ_j der Wert, um den die Dualvariablen y_i für $T_i \in \mathcal{V}_j$ erhöht werden, also

$$y_i = \sum_{j: T_i \in \mathcal{V}_j} \epsilon_j. \quad (4.10)$$

Der Wert der dualen Lösung lässt sich somit schreiben als

$$\sum_{i=1}^p y_i = \sum_{i=1}^p \sum_{j: T_i \in \mathcal{V}_j} \epsilon_j = \sum_{j=1}^{\ell} |\mathcal{V}_j| \epsilon_j, \quad (4.11)$$

wobei im zweiten Schritt wieder die Summationsreihenfolge vertauscht wurde. Genauso lassen sich die Kosten von A_f ausdrücken:

$$c(A_f) = \sum_{i=1}^p |A_f \cap T_i| y_i \stackrel{(4.10)}{=} \sum_{i=1}^p |A_f \cap T_i| \sum_{j: T_i \in \mathcal{V}_j} \epsilon_j = \sum_{j=1}^{\ell} \left(\sum_{T_i \in \mathcal{V}_j} |A_f \cap T_i| \right) \epsilon_j.$$

Gilt nun für den Ausdruck in der Klammer

$$\sum_{T_i \in \mathcal{V}_j} |A_f \cap T_i| \leq \gamma \cdot |\mathcal{V}_j| \quad (4.12)$$

so erhält man

$$c(A_f) \stackrel{(4.12)}{\leq} \sum_{j=1}^{\ell} \gamma \cdot |\mathcal{V}_j| \epsilon_j \stackrel{(4.11)}{=} \gamma \cdot \sum_{i=1}^p y_i \leq \gamma \cdot OPT .$$

Wegen des Rückwärtslösens darf A_f zu Beginn der j -ten Iteration durch eine minimale Erweiterung B ersetzt werden. Somit ist die Voraussetzung (4.9) lediglich die von einer speziellen Iteration unabhängig formulierte Forderung (4.12). Die Behauptung folgt. \square

Beispiel. Für jede Teillösung A des Problems MINIMAL SPANNENDER BAUM ist $\mathcal{V}(A)$ die Menge aller von Zusammenhangskomponenten induzierten Schnitte. Jede minimale Erweiterung B von A induziert einen spannenden Baum auf den zu Superknoten geschrumpften Zusammenhangskomponenten von A . Die Summe $\sum_{T_i \in \mathcal{V}(A)} |B \cap T_i|$ ist daher die Summe der Knotengrade eines Baumes mit $k = |\mathcal{V}(A)|$ Knoten, also $2k - 2$. Mit anderen Worten: Wir können in (4.9) $\gamma = 2$ wählen.

Satz 4.9 *Das Primal-duale Schema mit Synchronisierung ist ein 2-Approximationsalgorithmus für MINIMAL SPANNENDER BAUM.*

Leider können wir ohne weitere Kunstgriffe nicht garantieren, dass das Primal-duale Schema einen *optimalen* Spannbaum liefert: Die LP-Relaxation der HITTING SET Formulierung ist für dieses Problem zu schwach.

Bemerkung. Wir haben früher angemerkt, dass die relaxierten Bedingungen vom komplementären Schlupf nur im Durchschnitt erfüllt sein müssen. Obiges Beispiel präzisiert diese Bemerkung: Obwohl einzelne Superknoten einen hohen Grad haben können, ist der *durchschnittliche Knotengrad* eines Baums (allgemeiner: Waldes) höchstens zwei.

4.5 Allgemeine Anwendung im Netzwerk Design

Aus der letzten Bemerkung ergibt sich $\gamma = 2$ als „natürlicher Kandidat“ für einen Approximationsfaktor, wenn wir das Primal-duale Schema mit Synchronisierung anwenden. Wir wollen dies abschließend für eine allgemeine Klasse von NETZWERK DESIGN Problemen diskutieren und formulieren folgendes ganzzahliges Programm für einen Graphen $G = (V, E)$:

$$\begin{aligned} OPT = \min \quad & \sum_{e \in E} c_e x_e \\ & \sum_{e \in \delta(S)} x_e \geq f(S) \quad \emptyset \neq S \subseteq V \\ & x_e \in \{0, 1\} \quad e \in E , \end{aligned} \quad (4.13)$$

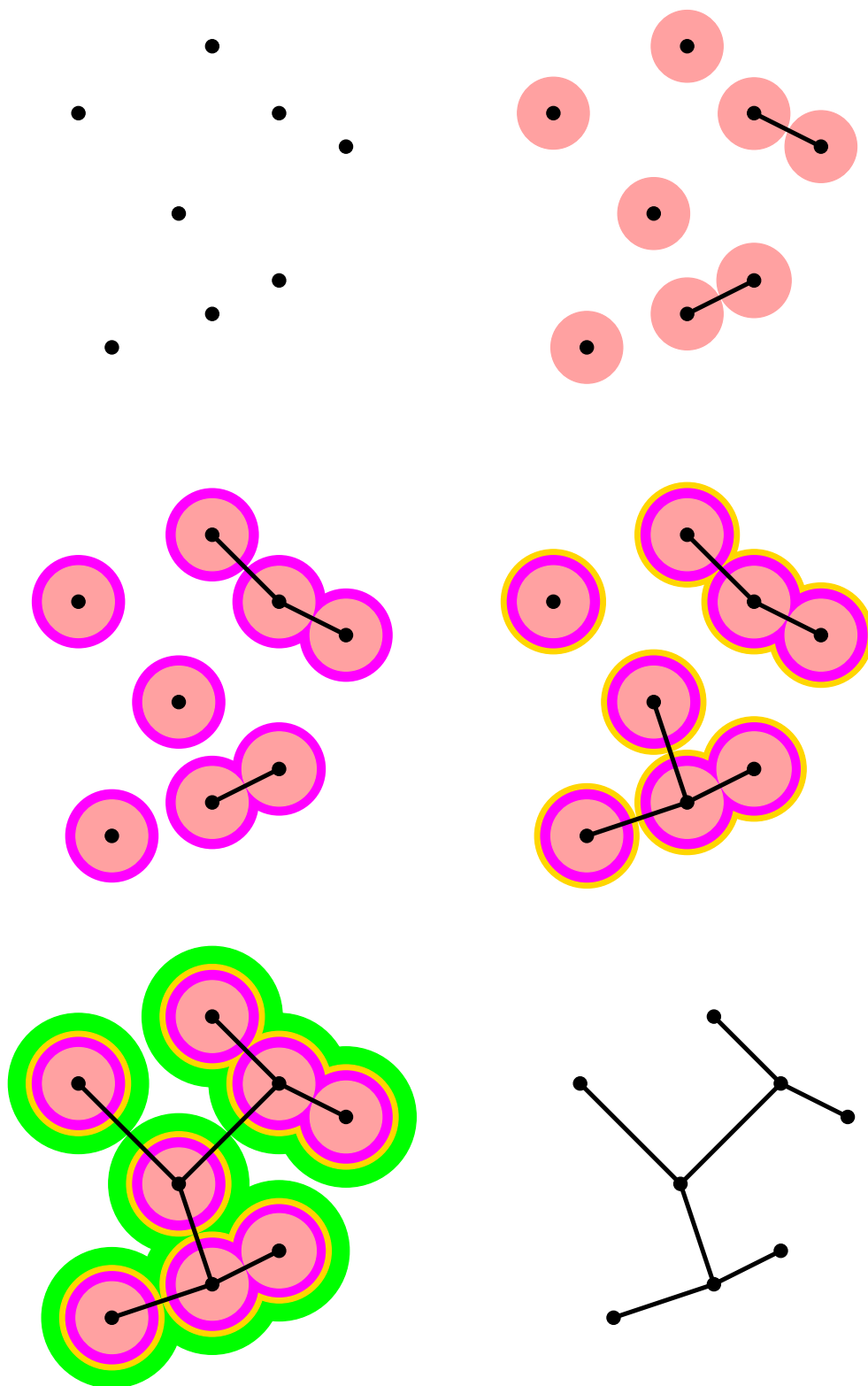


Abbildung 4.4: Veranschaulichung des simultanen Wachstums der Dualvariablen der von Zusammenhangskomponenten induzierten Schnitte als „Gräben“ für eine geometrische Instanz von MINIMAL SPANNENDER BAUM.

Hierbei sei $f : 2^V \rightarrow \{0, 1\}$ implizit gegeben, d.h. ein Orakel gibt zu gegebenem $S \subseteq V$ den Wert $f(S)$ zurück. (4.13) beschreibt ein HITTING SET Problem auf G , bei dem die gesuchte Kantenteilmenge $A \subseteq E$ alle Schnitte $\delta(S)$ mit $f(S) = 1$ „treffen“ muss.

Beispiel. Wählen wir in (4.13) $f(S) = 1$ für $\emptyset \neq S \subseteq V$ erhalten wir MINIMAL SPANNENDER BAUM. Haben wir zwei Knoten $s, t \in V$ ausgezeichnet, so beschreibt (4.13) mit $f(S) = 1$ für $S \subseteq V$ wobei $s \in S$ und $t \notin S$ das s - t -KÜRZESTER WEG Problem.

Der Vollständigkeit halber schreiben wir die LP-Relaxation von (4.13) auf:

$$\begin{aligned}
 LB = \min \quad & \sum_{e \in E} c_e x_e \\
 & \sum_{e \in \delta(S)} x_e \geq 1 \quad S : f(S) = 1 \\
 & x_e \geq 0 \quad e \in E,
 \end{aligned} \tag{4.14}$$

das hierzu duale lineare Programm lautet

$$\begin{aligned}
 LB = \max \quad & \sum_{S: f(S)=1} f(S) \cdot y_S \\
 & \sum_{S: e \in \delta(S)} y_S \leq c_e \quad e \in E \\
 & y_S \geq 0 \quad S : f(S) = 1.
 \end{aligned} \tag{4.15}$$

Eine wichtige Beobachtung ist, dass jede kantenminimale Lösung zu (4.13) ein Wald ist, denn aus jedem Kreis kann man eine beliebige Kante entfernen, ohne unzulässig zu werden. Wie zuvor nennen wir auch S verletzt, wenn $\delta(S)$ gemeint ist. Sei $\delta_A(S) = \delta(S) \cap A$. Dann ist S verletzt genau dann, wenn $\delta_A(S) = \emptyset$ obwohl $f(S) = 1$. Die Voraussetzung in Satz 4.8 liest sich nun

$$\sum_{S \in \mathcal{V}(A)} |\delta_B(S)| \leq \gamma |\mathcal{V}(A)| \tag{4.16}$$

für jede minimale Erweiterung B einer unzulässigen Teillösung A mit zugehörigen minimal verletzten Mengen $\mathcal{V}(A)$. Für allgemeine 0-1-wertige Funktionen f ist unbekannt, wie man (4.16) beweisen kann. Die folgende Funktionenklasse umfasst allerdings besonders viele interessante NETZWERK DESIGN Probleme.

Definition 4.10 Eine Funktion $f : 2^V \rightarrow \{0, 1\}$ heißt echt (engl. proper) falls

1. $f(V) = 0$
2. f maximal, d.h. für disjunkte $A, B \subseteq V$ gilt $f(A) = f(B) = 0 \Rightarrow f(A \cup B) = 0$
3. f symmetrisch, d.h. für alle $S \subseteq V$ gilt $f(S) = f(V \setminus S)$.

Minimal verletzte Mengen müssen keine so „übersichtliche“ Struktur haben wie wir es bisher gesehen haben. Zunächst ist weder klar, wie wie in der nun allgemeinen Situation die Zulässigkeit einer Teillösung überprüfen noch, wie eine oder alle verletzten Mengen effizient gefunden werden können. Man kann die Maximalität von f auch so auffassen: Gilt $f(S) = 1$, dann gilt für jede Partition von $S = S_1 \cup \dots \cup S_l$, dass $f(S_i) = 1$ für wenigstens ein $i \in \{1, \dots, l\}$. Das nutzen wir in folgendem Lemma aus.

Lemma 4.11 Sei $f : 2^V \rightarrow \{0,1\}$ maximal und $A \subseteq E$. Dann gilt

1. A ist zulässig für f genau dann, wenn für jede Zusammenhangskomponente C von (V, A) gilt $f(C) = 0$.
2. Die minimal verletzten Mengen von A sind genau die Zusammenhangskomponenten C von (V, A) für die $f(C) = 1$.

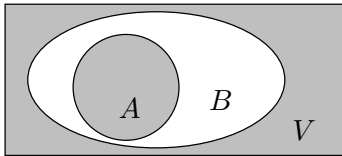
Beweis. Sei S eine verletzte Menge, d.h. $f(S) = 1$, aber $\delta_A(S) = \emptyset$. S kann keine Zusammenhangskomponente von (V, A) nur teilweise enthalten (dann wäre $\delta_A(S) \neq \emptyset$); S muss also die Vereinigung von Zusammenhangskomponenten sein. Weil aber f maximal ist, muss bereits $f(C) = 1$ für eine in S enthaltene Komponente C gelten: Es ist bereits C verletzt. Eine minimal verletzte Menge ist daher notwendigerweise eine Zusammenhangskomponente C von (V, A) und $f(C) = 1$ ist äquivalent mit ihrer Verletztheit. \square

last edit:
29.11.06

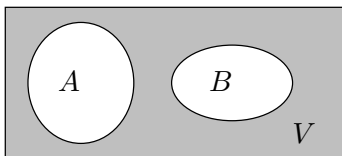
Selbst für maximale Funktionen f kann der Parameter γ in (4.16) beliebig groß werden [16]. Man muss wie z.B. in Definition 4.10 mehr fordern, etwa Symmetrie. Die Maximalität der Funktion f ist dann eine stärkere Eigenschaft, mit der wir anschließend ein allgemeines Approximationsresultat beweisen können.

Lemma 4.12 Sei $f : 2^V \rightarrow \{0,1\}$ symmetrisch. Dann ist f maximal genau dann wenn f komplementär ist, d.h. für alle $A \subseteq B \subseteq V$ mit $f(A) = f(B) = 0$ gilt $f(B \setminus A) = 0$.

Beweis. Sei f maximal und $A \subseteq B \subseteq V$ mit $f(A) = f(B) = 0$. Weil f symmetrisch ist, gilt $f(V \setminus B) = 0$. $V \setminus B$ und A sind disjunkt, f ist maximal, daher gilt $0 = f((V \setminus B) \cup A) = f(B \setminus A)$; die letzte Gleichheit ist Konsequenz der Symmetrie, d.h. f ist komplementär.



Sei umgekehrt f komplementär und es gelte $f(A) = f(B) = 0$ für $A, B \subseteq V$ mit $A \cap B = \emptyset$. Zu zeigen ist, dass f maximal ist, also $f(A \cup B) = 0$. Es ist $B \subseteq V \setminus A$. Weil f symmetrisch ist, gilt $f(V \setminus A) = 0$. Weil f komplementär ist, gilt $0 = f((V \setminus A) \setminus B) = f(A \cup B)$, wobei letztere Gleichheit wieder wegen der Symmetrie von f gilt.



\square

Satz 4.13 Das Primal-duale Schema mit Synchronisierung ist ein 2-Approximationsalgorithmus für das NETZWERK DESIGN Problem (4.13) für jede echte Funktion $f : 2^V \rightarrow \{0,1\}$.

Beweis. Zu zeigen ist (4.16) mit $\gamma = 2$. Wie eingangs erwähnt, interpretieren wir γ als Durchschnittsgrad eines Waldes. Weil f maximal ist, besteht $\mathcal{V}(A)$ zu gegebenem A nach

Lemma 4.11 genau aus den Zusammenhangskomponenten S von (V, A) mit $f(S) = 1$. Sei B eine minimale Erweiterung zu A . Wir konstruieren einen Graphen aus (V, B) , indem jede Zusammenhangskomponente des Graphen (V, A) zu einem Knoten geschrumpft wird. Bezeichne H die Knotenmenge des geschrumpften Graphen. Da B (kanten-)minimale Erweiterung ist (es gibt keine Kreise oder parallele Kanten nach dem Schrumpfen), können wir $B \setminus A$ mit der Kantenmenge F des geschrumpften Graphen identifizieren. Insbesondere ist (H, F) ein Wald.

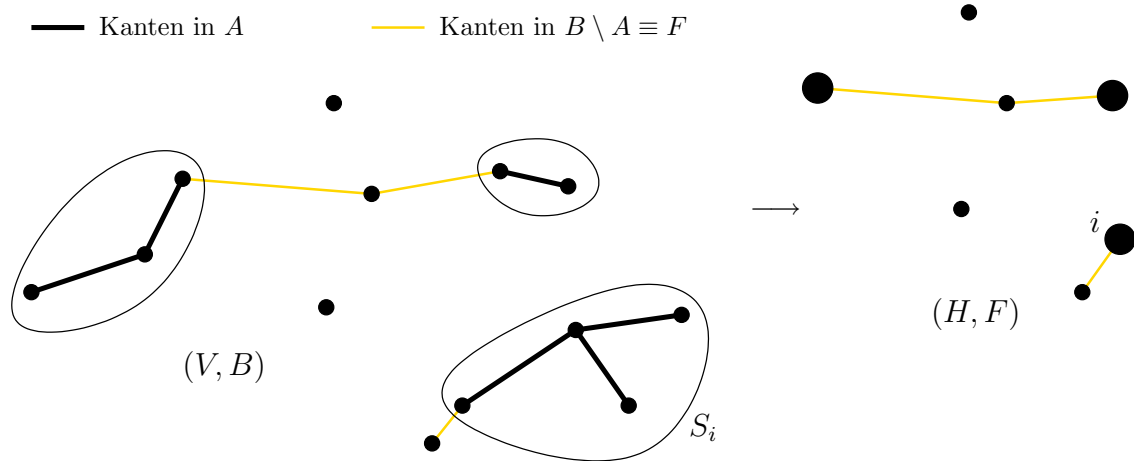


Abbildung 4.5: Schrumpfen von (V, B) im Beweis von Satz 4.13

Zu $i \in H$ bezeichne S_i die zugehörige Zusammenhangskomponente in (V, A) und sei $d_F(i)$ der Grad von i in (H, F) , d.h. $d_F(i) = |\delta_B(S_i)|$. Mit der Definition $W = \{i \in H \mid f(S_i) = 1\}$ schreiben wir die Menge der minimal verletzten Mengen als $\mathcal{V}(A) = \{S_i \mid i \in W\}$. Die zu zeigende Behauptung $\sum_{S \in \mathcal{V}(A)} |\delta_B(S)| \leq 2|\mathcal{V}(A)|$ können wir daher umschreiben zu

$$\sum_{i \in W} d_F(i) \leq 2|W| . \quad (4.17)$$

Wir benötigen noch ein weiteres Argument, um (4.17) zeigen zu können: Jedes Blatt $v \in H$ gehört zu W , d.h. es gibt keine Zusammenhangskomponente S_v in (V, A) mit $d_F(v) = 1$ (v ist ein Blatt) aber $f(S_v) = 0$ (S_v ist nicht verletzt). Nehmen wir an, dies gelte nicht für ein Blatt $v \in H$. Sei $e \in F$ die zu v inzidente Kante und sei C die Zusammenhangskomponente in (V, B) , die S_v enthält. Nun ist f komplementär, es gilt also wegen $f(S_v) = 0$ auch $f(C \setminus S_v) = 0$. Weil B minimale Erweiterung ist, ist $B \setminus \{e\}$ nicht zulässig, gleichbedeutend mit $f(S_v) = 1$ oder $f(C \setminus S_v) = 1$: Widerspruch. Die Annahme ist falsch; jedes Blatt $v \in H$ gehört zu W .

Entfernen wir isolierte Knoten aus H (für das Argument sind Knoten vom Grad 0 ohne Belang), so erhalten wir schließlich

$$\sum_{i \in W} d_F(i) = \sum_{i \in H} d_F(i) - \sum_{i \notin W} d_F(i) \leq (2|H| - 2) - 2(|H| - |W|) = 2|W| - 2 \quad (4.18)$$

sogar noch etwas stärker als gefordert. Die Ungleichung gilt, weil (H, F) ein Wald mit höchstens $|H| - 1$ kanten ist und alle Knoten nicht in W einen Grad von wenigstens 2 haben. \square

Bemerkung. Ist f in (4.13) eine echte Funktion, so kann man zeigen, dass eine minimale Erweiterung stets eindeutig ist. Die Reihenfolge des Löschens („Rückwärtslöschen“) redundanter Elemente dann nicht entscheidend.

Beispiel. Das STEINERWALD Problem verallgemeinert das STEINERBAUM Problem. Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit Kantengewichten $c : E \rightarrow \mathbb{Q}_+$ und eine Menge disjunkter Knotenmengen $T_1, \dots, T_p \subseteq V$. Gesucht ist ein Wald in G mit minimalen Kosten so, dass T_i , $i = 1, \dots, p$ jeweils zusammenhängend ist. Das ganzzahlige Programm (4.13) beschreibt dieses Problem, wenn $f(S) = 1$ genau dann gilt, wenn es ein $i \in \{1, \dots, p\}$ gibt mit $\emptyset \neq S \cap T_i \neq T_i$. Mit $p = 1$ ist das das STEINERBAUM Problem.

Die so definierte Funktion f ist echt: $f(V) = 0$ ist offenbar. Falls $f(A) = f(B) = 0$ für disjunkte $A, B \subseteq V$, so gilt $A \cap T_i = \emptyset$ oder $A \cap T_i = T_i$ für alle $i \in \{1, \dots, p\}$ (ebenso für B), und somit ist auch $(A \cup B) \cap T_i = \emptyset$ oder $(A \cup B) \cap T_i = T_i$ für alle $i \in \{1, \dots, p\}$; nach Definition von f gilt damit $f(A \cup B) = 0$, f ist also maximal. Außerdem gilt $A \cap T_i = \emptyset \Rightarrow (V \setminus A) \cap T_i = T_i$ und $A \cap T_i = T_i \Rightarrow (V \setminus A) \cap T_i = \emptyset$, somit ist f symmetrisch.

Satz 4.14 *Das Primal-duale Schema mit Synchronisierung ist ein 2-Approximationsalgorithmus für STEINERWALD (insbesondere für STEINERBAUM).*

Bemerkung. In diesem Kapitel haben wir nicht die Frage nach der *bestmöglichen* Approximationsgüte eines Problems gestellt. Etwa für MINIMUM MULTICUT auf Bäumen ist sie schwer zu beantworten: Das Problem betrachtet auf Sternen ist äquivalent zum VERTEX COVER Problem (s. Übung), so dass eine Lösung des offenen Problems 1.5 auch hier eine untere Schranke lieferte. Die Schlagkraft der vorgestellten Vorgehensweise (mit möglicherweise bewusstem Verzicht darauf, die *beste* Güte zu erzielen) liegt in der *sehr allgemeinen Anwendbarkeit* des Primal-dualen Schemas, nicht nur auf HITTING SET Probleme.

Kapitel 5

Aroras PTAS für Euklidisches TSP

last edit:
6.12.06

Bei geometrischen Optimierungsproblemen wird der Abstand $d(x, y)$ zweier Punkte x und y im d -dimensionalen Raum häufig in der Euklidischen Norm gemessen:

$$d(x, y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2} .$$

Dies schränkt geometrische Probleme weitaus stärker ein als entsprechende Probleme auf z.B. Graphen. Die Lage von n Punkten in der Ebene hat $2n$ Freiheitsgrade, während ein Problem auf einem Graph mit n Knoten $\binom{n}{2}$ voneinander unabhängige Distanzen haben kann. Gleichzeitig werden hierdurch häufig stärkere Approximationsaussagen ermöglicht als im allgemeinen Fall. Wir sehen uns in diesem Kapitel ein prominentes Beispiel an. Wir beweisen

Satz 5.1 *Es gibt ein PTAS für EUKLIDISCHES TSP.*

Bemerkung. Am Rand sei eine technische Kuriosität erwähnt. Beim Entscheidungsproblem zum EUKLIDISCHEN TSP fragt man, ob eine Tour der Länge höchstens k existiert. Die Tourlänge ist aber eine Summe von n Quadratwurzeln deren Vergleich mit k möglicherweise eine große arithmetische Präzision erfordert.

Offenes Problem 5.2 *Seien $a_1, \dots, a_n, k \in \mathbb{Z}$. Gibt es einen polynomialen Algorithmus, der entscheidet, ob $\sum_{i=1}^n \sqrt{a_i} \leq k$?*

Diese Schwierigkeit ist nicht spezifisch für schwere Probleme; sie tritt genauso bei leichten geometrischen Problemen auf. Sie ist allerdings dafür verantwortlich, dass man für die Entscheidungsversionen vieler geometrischer Optimierungsprobleme zwar zeigen kann, dass sie \mathcal{NP} -schwer sind, nicht aber, dass sie zu \mathcal{NP} gehören.

Bemerkung. Bei geometrischen Problemen wird häufig noch unformaler formuliert als ohnehin schon häufig in der Optimierung. Auch hier wird an die Intuition appelliert, wenn z.B. Touren bestimmte Gebiete „besuchen“, „betreten“ und „verlassen“.

5.1 Portale und Portaltouren

Polynomiale Approximationsschemata basieren oft auf der Beobachtung, dass sich \mathcal{NP} -schwere Probleme durch vollständige Enumeration in Polynomialzeit optimal „lösen“ lassen, wenn bestimmte Eingabegrößen *beschränkt* gehalten werden: Etwa der größte Profit der Gegenstände beim KNAPSACKPROBLEM oder die Anzahl verschiedener Größen der Gegenstände bei BIN PACKING. Die Kunst liegt darin, Einschränkungen (der Kosten, der Problemstruktur, ...) in Abhängigkeit von der vorgegebenen Genauigkeit ε so zu wählen, dass eine optimale eingeschränkte Lösung noch polynomial berechenbar ist, ihre Kosten sich gegenüber einer optimalen allgemeinen Lösung aber um nicht mehr als einen Faktor $1 + \varepsilon$ erhöhen.

Wir haben in Satz 2.1 gesehen, dass das allgemeine TRAVELING SALESMAN PROBLEM im Wesentlichen „überhaupt nicht“ zu approximieren ist. Für das METRISCHE TSP, d.h. wenn für die Kantengewichte die Dreiecksungleichung gilt, kennen wir den $\frac{3}{2}$ -Approximationsalgorithmus von Christofides. Andererseits weiß man: Es gibt keinen $220/219$ -Approximationsalgorithmus für das METRISCHE TSP, es sei denn $\mathcal{P} = \mathcal{NP}$ [33]. Man vermutet, dass $4/3$ der erreichbare Approximationsfaktor ist.

Offenes Problem 5.3 *Welcher Approximationsfaktor ist möglich für das METRISCHE TSP?*

Im Frühjahr 1996 wurden sehr überraschend von Arora [2] und Mitchell [30] unabhängig voneinander polynomiale Approximationsschemata für das EUKLIDISCHE TSP gefunden. Weil ersteres das besser bekannte ist (es lässt sich auf konstante Dimension $d \geq 2$ verallgemeinern), wird es hier beschrieben. Die oben erwähnten strukturellen Einschränkungen werden uns auf sogenannte Portaltouren führen, die wir mit dynamischer Programmierung auffinden können. Es handelt sich um eine typische Anwendung des *divide-and-conquer*-Prinzips: Das gesamte Problem wird rekursiv in kleinere Teilprobleme zerlegt, bis nur noch triviale Teilprobleme übrig bleiben. Teillösungen kleinerer Probleme werden jeweils bestmöglich zu Lösungen der nächst höheren Ebene in der Rekursion zusammen gesetzt.

Runden und Skalieren der Eingabedaten einer Instanz

Gegeben seien n Punkte in der Ebene mit ihren Koordinaten. Wir erzwingen zunächst eine gewisse *Ausgewogenheit* der Verteilung der Punkte. Sei L_0 die Seitenlänge des kleinsten alle Punkte umschließenden achsenparallelen Quadrats. Es gilt $2L_0 \leq OPT$. Wir überziehen das Quadrat mit Gitterlinien mit Abstand $L_0 \cdot \varepsilon / 4n$ und verschieben jeden Punkt auf den ihm am nächsten liegenden Gitterpunkt. Dabei fallen möglicherweise mehrere Punkte zu einem Punkt zusammen. Das Verschieben erhöht die Länge einer optimalen Tour „nur unwesentlich“, nämlich um höchstens $2n \cdot L_0 \cdot \varepsilon / 4n = \frac{1}{2}L_0 \cdot \varepsilon \leq \frac{1}{2}\varepsilon OPT$. Die Instanz wird nun skaliert, indem alle Distanzen mit $16n / (L_0 \cdot \varepsilon)$ multipliziert werden. Für die neuen Koordinaten der Punkte können wir daher ganzzahlige Vielfache von 4 annehmen, insbesondere ist der kleinste Abstand zwischen zwei Punkten mindestens 4 und der größte horizontale/vertikale Abstand zwischen zwei Punkten $4n$.

Für die Seitenlänge L eines alle Punkte umschließenden achsenparallelen Quadrats der skalierten Instanz können wir $L = 4n \cdot n = 4n^2$ wählen. Schließlich sei für die Darstellung hier ohne Einschränkung die Anzahl der Punkte n eine Zweierpotenz (das kann durch iden-

tische Kopien vorhandener Punkte erreicht werden und verdoppelt höchstens die Anzahl der Punkte), also $L = 2^k$ mit $k = 2 + 2 \log_2 n$.

Rekursive Partitionierung der Punkte

Wir partitionieren Quadrate rekursiv in vier gleich große Teile, also das erste $L \times L$ Quadrat in vier $L/2 \times L/2$ Quadrate usw. Für unsere grobe Analyse partitionieren wir bis Einheitsquadrate entstehen. Für eine bessere Analyse partitioniert man, bis jedes Quadrat höchstens noch einen Punkt enthält, vergleiche den Unterschied in Abbildung 5.1. Interpretiert man jedes entstehende Quadrat als einen Knoten, kann man die Partitionierung in einem Baum T kodieren, einem sogenannten *quadtrees*. Die Tiefe eines Knoten in T nennen wir die Tiefe des zugehörigen Quadrats; das Ausgangsquadrat erhält Tiefe 0. Somit haben Quadrate der Tiefe i eine Seitenlänge von $L/2^i$. Die Tiefe des Baumes ist $k = O(\log n)$.¹

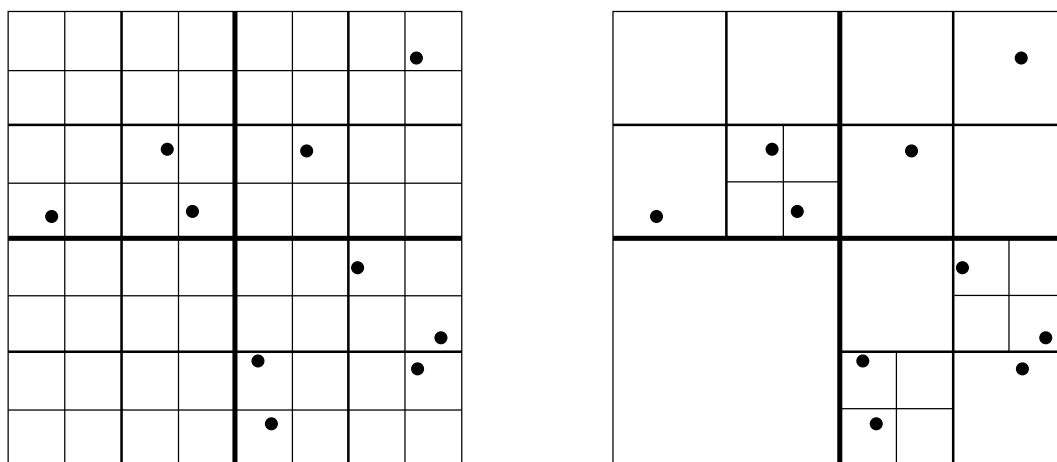


Abbildung 5.1: Rekursive Partition der TSP-Instanz bis zur Tiefe 3 und zugehöriger *quadtrees*

Entsprechend definieren wir die Tiefe der Linien(-segmente), die die Partitionierung erzeugen. Die beiden Linien, die das Ausgangsquadrat zerteilen erhalten Tiefe 1, allgemein erhalten die 2^i Linien, die die Quadrate der Tiefe $i - 1$ in Quadrate der Tiefe i zerteilen die Tiefe i . Die längste Seitenlänge eines Quadrats, das eine Linie der Tiefe i berührt ist demnach $L/2^i$. Wir dürfen annehmen, dass in der skalierten Instanz kein Punkt auf dem Rand eines Quadrats liegt (etwa indem Linien halbzahlige Koordinaten erhalten).

Portale

Auf jeder Linie zeichnen wir äquidistante Punkte aus, sogenannte *Portale*. Die approximative Tour, die wir konstruieren, darf Linien nur an Portalen kreuzen. Intuitiv darf es nicht zu viele Portale geben, um die rekursive Konstruktion einer Tour in Polynomialzeit zu gewährleisten, andererseits müssen genügend Portale vorhanden sein, um die erzwungenen Umwege kurz zu halten.

¹Die O -Notation tut an dieser Stelle weh, weil wir k genau angeben können. Gewöhnen wir uns besser gleich an die Schmerzen; in diesem Kapitel wird noch so manche Abschätzung wehtun...

Auf einer Linie der Tiefe i sei der Abstand zwischen zwei Portalen $L/(2^i m)$, wobei der *Portalparameter* m eine Zweierpotenz aus dem Intervall $[k/\varepsilon, 2k/\varepsilon]$ ist. Bei dieser Wahl ist $m = O(\log n/\varepsilon)$. Weil $L/2^i$ die längste Seitenlänge eines Quadrats der Tiefe i ist, hat jedes Quadrat höchstens $(L/2^i)/(L/2^i m) = m$ Portale an jeder Seite, also höchstens $4m$ Portale insgesamt. Man beachte, dass mit wachsender geforderter Genauigkeit, d.h. kleinerem ε ebenso die Anzahl der Portale wächst. Weil m als Zweierpotenz gewählt wurde, ist jedes Portal eines Quadrats Q der Tiefe i auch Portal aller derer Quadrate mit geringerer Tiefe als i , die mit Q gemeinsame Seiten auf einer Linie haben.

Definition 5.4 Eine Portaltour (engl. portal respecting tour) besucht alle n Knoten und eine Teilmenge der Portale. Kreuzungen der Tour mit Linien müssen an Portalen stattfinden, Portale dürfen mehrfach besucht werden, aber eine Portaltour kreuzt sich nicht selbst.

5.2 Dynamisches Programm

Wir werden nun solche Portaltouren konstruieren, die jedes Quadrat nur beschränkt häufig betreten oder verlassen. Das Teilproblem innerhalb eines Quadrats ist unabhängig von seinem Äußeren zu lösen, wenn bekannt ist, welche Portale in welcher Reihenfolge von der Portaltour benutzt werden. Arora nennt diese Information das *Interface*, also die Schnittstelle. Es muss dann innerhalb eines Quadrats keine Tour, sondern knotendisjunkte Wege gefunden werden, die alle Knoten im Inneren besuchen und mit der Schnittstelleninformation konsistent sind, s. Abbildung 5.2. Man beachte, dass wir hier erlauben, dass Quadrate gar keine Punkte enthalten. Das dynamische Programm wird eine Tabelle anlegen. Sie enthält je einen Eintrag mit den Kosten optimaler Wege im Inneren für *jedes* Quadrat und *jede* mögliche Schnittstelleninformation.

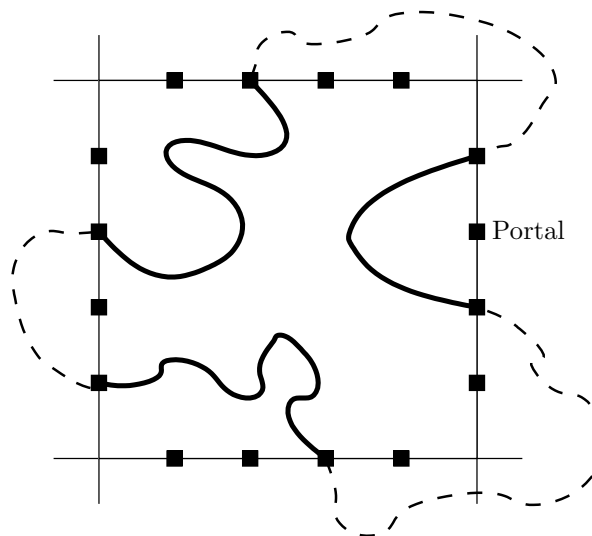


Abbildung 5.2: Knotendisjunkte Wege im Inneren eines Quadrats

Lemma 5.5 Zu jeder Portaltour gibt es eine Portaltour mit nicht höheren Kosten so, das kein Portal mehr als zweimal besucht wird.

Beweis. Wegen der Gültigkeit der Dreiecksungleichung führt das „Abkürzen“ weiterer Besuche zu keiner Erhöhung der Kosten, s. Abbildung 5.3. Sollte es hierdurch zu einer Selbstüberschneidung der Tour kommen, so liefert auch das Entkreuzen eine nicht teurere Tour. Es entstehen keine Subtours, d.h. mehr als eine Zusammenhangskomponente der Tour, wenn an einem Portal nicht mehr abgekürzt wird, wenn weniger als drei Besuche verbleiben. Nehmen wir das Gegenteil an: Es gibt eine Portaltour, die ein gegebenes Portal p wenigstens dreimal besucht, so dass jede Abkürzung an diesem Portal zu einer Subtour führt. Seien v_1, \dots, v_l , $l \geq 3$, die mit p verbundenen Punkte. Führt jede Kante (v_i, v_j) mit $1 \leq i \neq j \leq l$ zu einer Subtour, müssen bereits alle diese Paare (v_i, v_j) jeweils über einen Weg miteinander verbunden sein, : Ein Widerspruch zu der Bedingung, dass jedes v_i Knotengrad 2 hat. \square

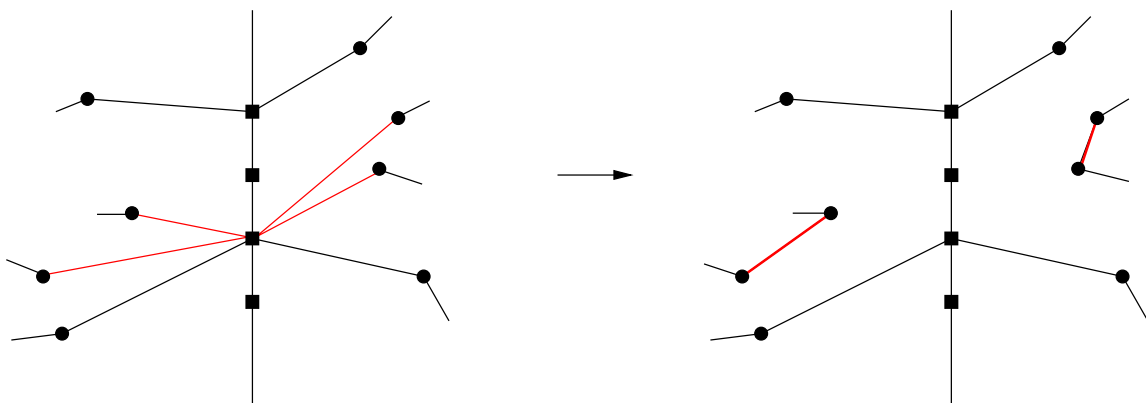


Abbildung 5.3: „Abkürzen“ mehrfacher Besuche eines Portals

Lemma 5.6 Eine optimale Portaltour kann mit Aufwand $2^{O(m)} = n^{O(1/\varepsilon)}$ berechnet werden.

last edit:
12.12.06

Beweis. Aufgrund von Lemma 5.5 können wir uns auf Portaltouren beschränken, die jedes Portal höchstens zweimal besuchen. Bezeichne mit τ eine solche zu konstruierende optimale Portaltour. Maximal besucht τ die $4m$ Portale eines gegebenen Quadrats Q in der Summe $8m$ Mal. Der Teil von τ innerhalb von Q besteht also aus höchstens $4m$ knotendisjunkten Wegen, die Q jeweils genau einmal betreten und genau einmal verlassen, so dass jeder Punkt innerhalb von Q auf genau einem dieser Wege liegt. Nicht alle Möglichkeiten führen auf überschneidungsfreie Touren. Tatsächlich müssen sich Touren wie *balancierte Klammern* verhalten, s. Abbildung 5.4.

Wieviele verschiedene *gültige Besuche* gibt es höchstens innerhalb von Q ? Jedes Portal wird 0, 1 oder 2 Mal besucht, insgesamt ergibt das höchstens $3^{4m} = n^{O(1/\varepsilon)}$ Möglichkeiten. Unter diesen betrachte nur diejenigen, die auch zu gültigen Touren gehören können, also eine gerade Anzahl von Ein- und Austritten von Q haben. Für eine feste solche Wahl, die $2r$ Portale besucht, ist die Anzahl der erlaubten Portal-Paare (d.h. der balancierten Klammern) gerade die r -te *Katalanzahl* $C_r = 1/(r+1) \cdot \binom{2r}{r} \leq 2^{2r}$. Für die Ungleichung rechnet man nach, dass $C_{r+1} \leq 4C_r$. Nun ist wiederum $2^{2r} = n^{O(1/\varepsilon)}$. Die Anzahl der gültigen Besuche innerhalb eines Quadrats sind also insgesamt $n^{O(1/\varepsilon)} \cdot n^{O(1/\varepsilon)} = n^{O(1/\varepsilon)}$. Weil es höchstens $L^2 + \frac{1}{4}L^2 + \frac{1}{16}L^2 + \dots + 1 \leq 2L^2 = O(n^4)$ Quadrate gibt, ist die Anzahl der Tabelleneinträge ebenfalls durch $n^{O(1/\varepsilon)}$ beschränkt.

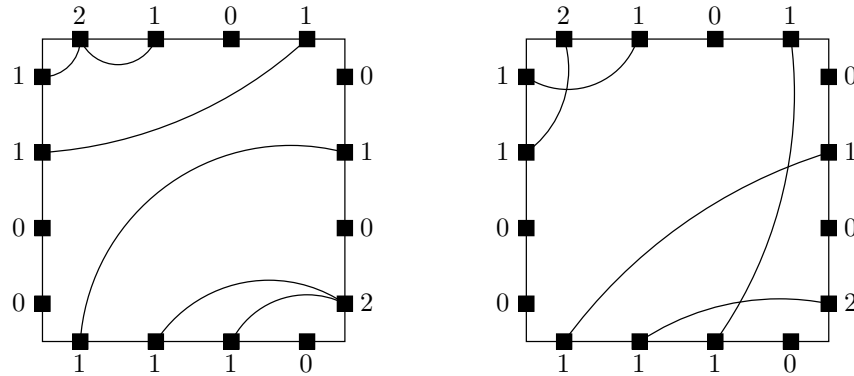


Abbildung 5.4: Gültige und ungültige Paare von Ein- und Austritten („Klammerungen“)

Für jeden Tabelleneintrag muss die optimale Länge eines gültigen Besuchs berechnet werden. Blätter enthalten höchstens einen Punkt; die Kosten der optimalen Wege in einem Blatt zu anzahl- und klammerungsmäßig gegebenen $2r$ Portalbenutzungen werden durch versuchsweises Einfügen des höchstens einen Punkts in alle $r = O(\log n/\varepsilon)$ Wege bestimmt. Betrachten wir nun ein beliebiges Quadrat Q der Tiefe i ; Portalbenutzungen seien anzahl- und klammerungsmäßig gegeben. Q hat vier Kinder der Tiefe $i + 1$ mit insgesamt weiteren $4m$ Portalen im Inneren von Q . Wieder wird jedes dieser Portale höchstens zweimal benutzt, was wie oben höchstens $3^{4m} = n^{O(1/\varepsilon)}$ Möglichkeiten ergibt. Sei hieraus eine Nutzung der Portale fest gewählt. Für jedes der vier Quadrate in Tiefe $i + 1$ muss jede überschneidungsfreie Paarung der jeweils höchstens $m/2 + m/2 + m + m = 3m$ Portale betrachtet werden. Deren Anzahl kann wie oben durch die Katalanzahl C_r mit $2r \leq 3m$ beschränkt werden, ist also $n^{O(1/\varepsilon)}$. Für jede gültige zahlenmäßige Auswahl und Klammerung von Portalen der Quadrate der Tiefe $i + 1$ haben wir die optimalen Kosten eines gültigen Besuchs bereits in der Rekursionsebene tiefer berechnet. Die kostenminimale Summe der vier gültigen Besuche in Tiefe $i + 1$ ergibt die Kosten eines optimalen gültigen Besuchs in Q . \square

Bemerkung. Beim Zusammenfügen vierer Quadrate zu einem Quadrat auf der nächsthöheren Rekursionsebene ist die Information der Portalbenutzungen und Klammerungen verfügbar, so dass alle Besuche ausgefiltert werden können (und müssen), die im Inneren Subtours erzeugten.

Bemerkung. Das dynamische Programm berechnet die Kosten einer optimalen Portaltour, nicht die Tour selbst. Diese muss aus den in der Tabelle hinterlegten Entscheidungen rekonstruiert werden.

5.3 Randomisiert verschobene Partitionierung

Lemma 5.6 versetzt uns zwar in die Lage, eine optimale Portaltour in Polynomialzeit zu bestimmen. Wir müssen allerdings noch zeigen, dass solch eine Tour eine Länge von höchstens $(1 + \varepsilon) \cdot OPT$ hat. Interessanterweise können wir das im Allgemeinen nicht, s. Abbildung 5.5. Wenn eine optimale Tour Linien geringer Tiefe, d.h. mit größerem Portalabstand häufig

kreuzt, kann die dargestellte Situation auftreten. Die folgende Konstruktion nutzt aus, dass es von den Linien geringer Tiefe nur relativ wenige gibt.

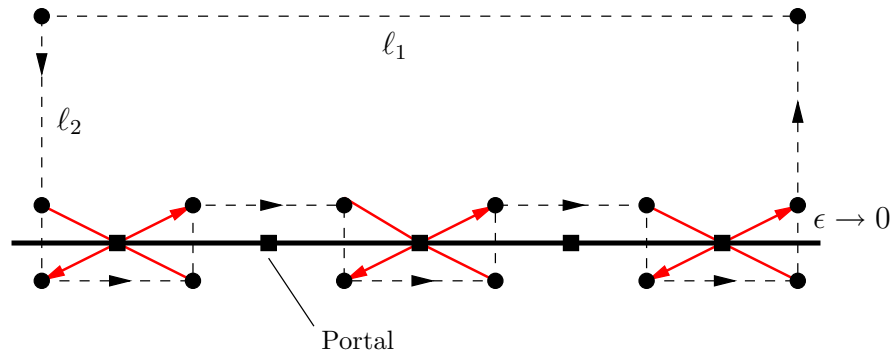


Abbildung 5.5: Umweg um einem konstanten Anteil an OPT bei Benutzung von Portalen. Für ϵ nahe 0 hat die optimale Tour (gestrichelt) eine Länge von etwa $2(\ell_1 + \ell_2)$. Der rot (durchgehend) eingezeichnete Umweg durch die Portale verlängert die Tour um etwa ℓ_1 .

Verschieben der Partitionierung

Seien $0 \leq a, b < L$ ganze Zahlen. Wir betrachten die sogenannte (a, b) -Partitionierung: Jede vertikale Linie an der Position x wird nach $a + x \bmod L$ verschoben; jede horizontale Linie an der Position y wird nach $b + y \bmod L$ verschoben. Die Portale werden mit verschoben, die Koordinaten der Punkte der TSP-Instanz bleiben unverändert. Wie Abbildung 5.6 suggeriert, werden Quadrate als über den Rand des Ausgangsquadrats „hinausgeschoben“ interpretiert. Diese Verschiebung verringert höchstens die Anzahl der Tabelleneinträge unseres dynamischen Programms, weil Touren den Rand nicht übertreten können.

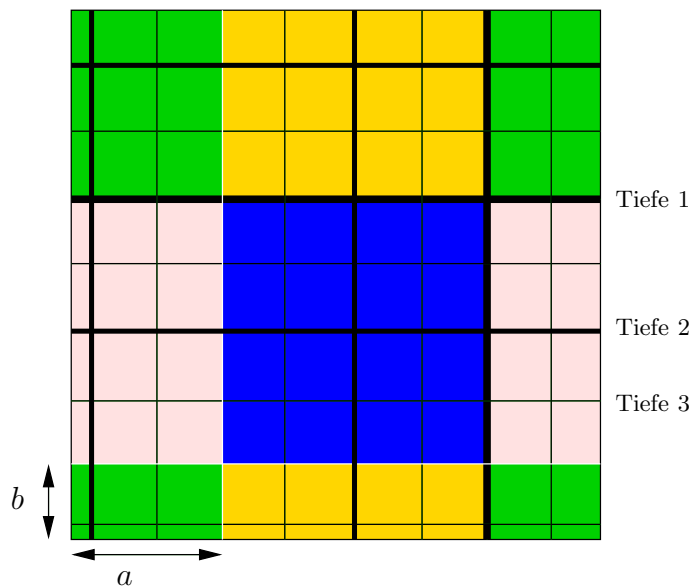


Abbildung 5.6: (a, b) -Partitionierung: Verschieben der Linien

Lemma 5.7 Sei π eine optimale TSP-Tour einer skalierten Instanz und $n(\pi)$ die Anzahl, wie oft π horizontale oder vertikale Linien des Einheitsgitters kreuzt. Es gilt

$$n(\pi) \leq 2 \cdot OPT .$$

Beweis. Sei s die Länge einer Kante e in π , u und v die Längen der Projektionen von s auf die Koordinatenachsen, also $s^2 = u^2 + v^2$. Die Kante e hat daher höchstens $u + 1 + v + 1$ Schnittpunkte mit den Einheitsgitterlinien. Nun ist

$$u + v + 2 \leq \sqrt{2(u^2 + v^2)} + 2 = \sqrt{2s^2} + 2 \leq 2s ,$$

wobei die erste Ungleichung wegen $(u+v)^2 \leq 2(u^2+v^2)$ gilt und wir in der zweiten Ungleichung wegen unserer eingänglichen Rundung $s \geq 4$ annehmen dürfen. \square

Satz 5.8 Seien ganzzahlige a, b zufällig (unabhängig und gleichverteilt) aus $[0, L]$ gewählt. Überführen wir eine optimale TSP-Tour in eine Portaltour der (a, b) -Partitionierung, die jedes Portal höchstens zweimal besucht, ist die erwartete Kostenerhöhung höchstens $2\varepsilon \cdot OPT$.

Beweis. Jede Kante (u, v) der optimalen Tour π , die eine Linie l nicht an einem Portal, sondern in einem Nichtportal q kreuzt, muss durch zwei Segmente (u, p) , (p, v) ersetzt werden, wobei p ein nächstes an q gelegenes Portal ist. Dies erhöht die Kosten um höchstens die Distanz zwischen zwei Portalen auf der Linie l .

Weil a, b gleichverteilt aus $[0, L]$ gewählt wurden, ist die Wahrscheinlichkeit, dass l eine Linie der Tiefe i ist gerade $2^i/L$: Die Anzahl der Linien der Tiefe i geteilt durch die Anzahl aller Linien. Auf einer Linie der Tiefe i ist die Distanz zwischen zwei Portalen $L/(2^i m)$. Der Erwartungswert der Kostenerhöhung beim Erzwingen der Benutzung eines Portals für eine Kreuzung mit einer Linie ist daher

$$\sum_{i=1}^k \frac{2^i}{L} \cdot \frac{L}{2^i m} = \frac{k}{m} \leq \varepsilon ,$$

weil $m \in [k/\varepsilon, 2k/\varepsilon]$. Zusammen mit Lemma 5.7 ergibt sich als erwartete Kostenerhöhung für alle Kreuzungen höchstens $2\varepsilon \cdot OPT$. \square

Korollar 5.9 Seien ganzzahlige a, b zufällig (unabhängig und gleichverteilt) aus $[0, L]$ gewählt. Dann gibt es mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ eine Portaltour der Länge höchstens $(1 + 4\varepsilon) \cdot OPT$, die jedes Portal der (a, b) -Partitionierung höchstens zweimal besucht.

Beweis. Folgt unmittelbar aus der Markov-Ungleichung (Lemma 3.9):

$$\Pr[\text{Kostenerhöhung ist größer als } 4\varepsilon \cdot OPT] \leq 2\varepsilon \cdot OPT / 4\varepsilon \cdot OPT = \frac{1}{2}$$

\square

Derandomisierung

Die einzige zufällige Entscheidung im beschriebenen randomisierten PTAS ist die Verschiebung der Partitionierung. Wählt man deterministisch unter allen (a, b) -Partitionierungen diejenige, die auf geringste Kosten der entstehenden Portaltour führt, erhöht sich die Laufzeit des PTAS um einen Faktor $L^2 = O(n^2)$.

Bemerkung. Das Approximationsschema benutzt wenig Struktur des TSP, sondern nutzt vielmehr die geometrische Natur des Problems aus. Das PTAS kann daher auf andere geometrische Probleme angewendet werden [2].

Bemerkung. Arora verbessert die Analyse des Algorithmus (u.a. durch Benutzung lediglich des *quadtrees*) auf $O(n(\log n)^{O(1/\varepsilon)})$, also „fast Linearzeit“, wie es im Titel seiner Arbeit heißt. Trotzdem sind die versteckten Konstanten sehr groß und die Frage nach der Praktikabilität eines PTAS ist legitim. Man kann die Existenz eines PTAS als Gutartigkeit eines Problems werten, das sich in der Praxis etwa mit guten Heuristiken zufrieden stellend lösen lässt.

Fazit

Man kann in Bezug auf die algorithmische Eleganz eines Approximationsschemas sicherlich geteilter Meinung sein. Die Idee, die Struktur einer Lösung geschickt einzuschränken ist herausragend, der resultierende Algorithmus häufig leider nur die vollständige Enumeration aller in Frage kommenden Lösungen mit anschließender Auswahl der günstigsten. Es gibt allerdings auch für geometrische Optimierungsprobleme „schöne“ Approximationsalgorithmen [9].

Kapitel 6

Semidefinite Relaxation

last edit:
13.12.06

Mit randomisiertem Runden haben wir bereits früher Approximationsresultate zeigen können. In diesem Kapitel lernen wir eine ausgereifere Variante kennen. In der Hauptsache wird es aber darum gehen, eine weitere allgemeine Technik vorzustellen, mit der sich untere Schranken für OPT gewinnen lassen: Semidefinite Relaxationen. Goemans und Williamson [15] haben diese Idee im Zusammenhang mit Approximationsalgorithmen 1994 erstmals verwendet.

Ein Maximierungsproblem

Beim MAX CUT Problem sind ein ungerichteter Graph $G = (V, E)$ sowie nichtnegative Kantengewichte $w : E \rightarrow \mathbb{Q}_+$ gegeben. Für $ij \notin E$ vereinbaren wir $w_{ij} = 0$. Gesucht ist ein Schnitt maximalen Gewichts, d.h. $S \subseteq V$ so, dass $w(\delta(S)) = \sum_{ij \in \delta(S)} w_{ij}$ maximal. Eine offensichtliche obere Schranke für OPT ist $\sum_{i < j} w_{ij}$, bzw. $|E|$ im ungewichteten Fall. Etwa für einen Sterngraph wird diese Schranke auch angenommen.

Satz 6.1 (Håstad [20]) Für $\varepsilon > 0$ gibt es keinen $(16/17 + \varepsilon)$ -Approximationsalgorithmus für MAX CUT, es sei denn $\mathcal{P} = \mathcal{NP}$.

Randomisiertes Runden: Der naive Ansatz

Die Partition $S, V \setminus S$ ist eine 2-Färbung der Knoten. Entscheiden wir für jeden Knoten mit Wahrscheinlichkeit je 1/2, ob er weiß oder schwarz gefärbt wird, erhalten wir einen Approximationsalgorithmus.

Algorithmus 11 Randomisiertes Runden für MAX CUT

```
 $S = \emptyset$ 
for  $i = 1, \dots, n$  do
  Wirf eine faire Münze
  if Münze zeigt Kopf then
     $S = S \cup \{i\}$ 
  end if
end for
```

Satz 6.2 *Randomisiertes Runden für MAX CUT ist ein 1/2-Approximationsalgorithmus.*

Beweis. Definiere für alle $ij \in E$ eine Zufallsvariable X_{ij} mit

$$X_{ij} = \begin{cases} 1 & \text{falls } ij \in \delta(S) \\ 0 & \text{sonst} \end{cases}$$

Es sei $W = \sum_{i,j:i < j} w_{ij} X_{ij}$ der Wert eines Schnittes nach randomisiertem Runden. Wir haben $\Pr[ij \in \delta(S)] = 1/2$, weil dieses Ereignis in genau den zwei (von vier) Fällen eintritt, in denen i und j unterschiedliche Farben haben. Für den Erwartungswert von W gilt

$$\begin{aligned} \mathbf{E}[W] &= \mathbf{E}\left[\sum_{i < j} w_{ij} X_{ij}\right] \\ &= \sum_{i < j} w_{ij} \mathbf{E}[X_{ij}] \\ &= \sum_{i < j} w_{ij} \Pr[ij \in \delta(S)] \\ &= \frac{1}{2} \sum_{i < j} w_{ij} \\ &\geq \frac{1}{2} OPT, \end{aligned}$$

wobei in der Ungleichung die obere Schranke $\sum_{i < j} w_{ij} \geq OPT$ benutzt wurde. □

Es ist keine Formulierung von MAX CUT als ganzzahliges Programm bekannt, deren LP-Relaxation eine Ganzzahligkeitslücke besser als 1/2 hat. Will man Satz 6.2 verbessern, muss $\Pr[ij \in \delta(S)]$ über 1/2 erhöht werden. Für die entscheidende Idee müssen wir einige Begriffe bereitstellen.

Semidefinite Programme

Definition 6.3 *Eine Matrix $X \in \mathbb{R}^{n \times n}$ heißt positiv semidefinit genau dann, wenn für alle $\mathbf{y} \in \mathbb{R}^n$ gilt $\mathbf{y}^T X \mathbf{y} \geq 0$. Man schreibt auch $X \succeq 0$.*

Lemma 6.4 *Ist X symmetrisch, dann sind äquivalent:*

- X ist positiv semidefinit
- X hat nicht-negative Eigenwerte
- $X = Y^T Y$ für eine Matrix $Y \in \mathbb{R}^{m \times n}$ mit $m \leq n$

Bemerkung. Jede symmetrische Matrix X kann mit Hilfe der *Cholesky-Zerlegung* als $Y^T Y$ unter Verwendung exakter Arithmetik in $O(n^3)$ faktorisiert werden, wobei die Berechnung Quadratwurzelziehen benötigt. Man bricht die Faktorisierung daher nach Erreichen der gewünschten Genauigkeit ab; der entstehende Fehler ist innerhalb der Numerik sehr gut untersucht und beherrschbar. Für Details siehe etwa das Buch von Golub und van Loan [17].

Ein *semidefinites Programm* ist

$$\begin{aligned} \max \text{ oder } \min \quad & \sum_{i,j} c_{ij} x_{ij} \\ & \sum_{i,j} a_{ijk} x_{ij} = b_k \\ & X = (x_{ij}) \succeq 0 \quad \text{und } X \text{ symmetrisch} \end{aligned} \tag{6.1}$$

Semidefinite Programme können für jedes $\varepsilon > 0$ bis auf einen additiven Fehler von ε optimal in Zeit polynomial in n und $\log(1/\varepsilon)$ gelöst werden (Ellipsoid-Methode, Innere-Punkte-Verfahren und weitere). Der Fehler ist, wie bei der Cholesky-Zerlegung, notwendig, weil die Lösung irrational sein kann. Semidefinite Programme sind in ihrer Theorie den linearen Programmen sehr ähnlich, insbesondere gilt auch hier eine starke Dualität; sie führen aber allgemein auf stärkere Relaxationen, d.h. bessere Schranken.

Das semidefinite Programm (6.1) ist äquivalent zu einem sogenannten *Vektorprogramm*

$$\begin{aligned} \max \text{ oder } \min \quad & \sum_{i,j} c_{ij} (\mathbf{y}_i \cdot \mathbf{y}_j) \\ & \sum_{i,j} a_{ijk} (\mathbf{y}_i \cdot \mathbf{y}_j) = b_k \\ & \mathbf{y}_i \in \mathbb{R}^m \quad i \in V \end{aligned} \tag{6.2}$$

Die Äquivalenz sehen wir mit der dritten Eigenschaft in Lemma 6.4. Sei $Y = (\mathbf{y}_1 \dots \mathbf{y}_n)$, d.h. die i -te Spalte von Y besteht aus \mathbf{y}_i . Dann ist genau $x_{ij} = \mathbf{y}_i \cdot \mathbf{y}_j$ ($= \mathbf{y}_i^T \mathbf{y}_j$, aber wir könnten auch andere Skalarprodukte zulassen). Eine zulässige Lösung für (6.2) gibt eine zulässige Lösung für (6.1) mit demselben Wert und umgekehrt.

Ein quadratisches ganzzahliges Programm

$$\begin{aligned} OPT = \max \quad & \frac{1}{2} \sum_{i < j} w_{ij} (1 - y_i y_j) \\ & y_i \in \{-1, 1\} \quad i \in V \end{aligned} \tag{6.3}$$

Dieses Programm modelliert MAX CUT: Für eine Kante $ij \in E$ ist $(1 - y_i y_j) = 0$, falls $i, j \in S$ und $(1 - y_i y_j) = 2$ sonst. Insbesondere induziert die Menge $S = \{i \in V \mid y_i = 1\}$ einen Schnitt.

Wir betrachten nun eine interessante Relaxation. Dazu fassen wir y_i als einen eindimensionalen Einheitsvektor auf. Die Relaxation besteht darin, Einheitsvektoren \mathbf{y}_i höherer Dimension $m \leq n$ zuzulassen, d.h. $\mathbf{y}_i \cdot \mathbf{y}_i = 1$, s. Abbildung 6.1. Wir erhalten

$$\begin{aligned} UB = \max \quad & \frac{1}{2} \sum_{i < j} w_{ij} (1 - \mathbf{y}_i \cdot \mathbf{y}_j) \\ & \mathbf{y}_i \cdot \mathbf{y}_i = 1 \quad i \in V \\ & \mathbf{y}_i \in \mathbb{R}^m \quad i \in V \end{aligned} \tag{6.4}$$

Das ist ein Vektorprogramm, das wir (bis auf einen additiven Fehler) in Polynomialzeit lösen können. Die Dimension m ist hierbei der (unbekannte) Zeilenrang der Matrix Y aus der Cholesky-Zerlegung und wir wählen $m = n$. Um aus einer optimalen Lösung für (6.4) eine zulässige Lösung für (6.3) zu erhalten, wenden wir eine uns schon bekannte Technik an: Randomisiertes Runden.

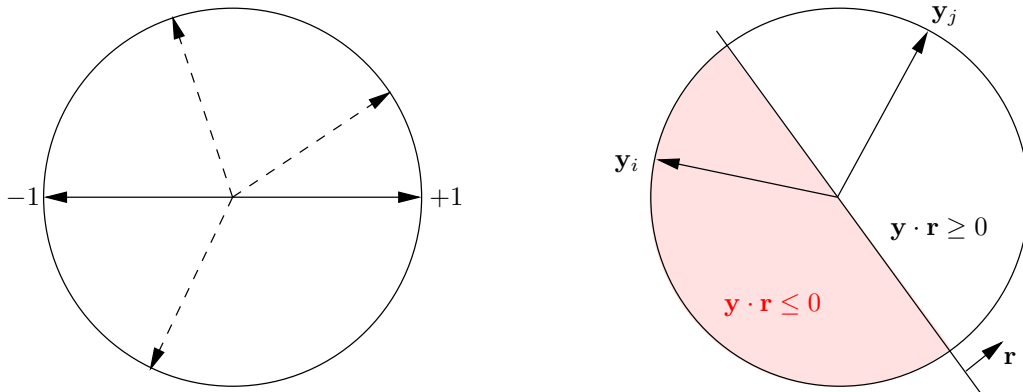


Abbildung 6.1: Relaxation eines eindimensionalen Einheitsvektors zu einem höherdimensionalen Einheitsvektor. Rechts ist die Lage der zufällig gewählten Hyperebene im Verhältnis zu den Einheitsvektoren der Lösung des Vektorprogramms (6.4) dargestellt.

Algorithmus 12 MAX CUT

Löse das Vektorprogramm (6.4); sei \mathbf{y}^* eine optimale Lösung
 Wähle zufällig gleichverteilt einen m -dimensionalen Einheitsvektor \mathbf{r}
 $S = \emptyset$
for $i = 1, \dots, n$ **do**
 if $\mathbf{y}_i \cdot \mathbf{r} \geq 0$ **then**
 $S = S \cup \{i\}$
 end if
end for

Der zufällig gewählte Vektor \mathbf{r} ist Normale einer Hyperebene durch den Ursprung. Alle Vektoren mit nicht-negativem Skalarprodukt mit \mathbf{r} befinden sich auf der einen Seite der Hyperebene, alle anderen Vektoren auf der anderen. Im Gegensatz zum Münzwurf ist es um die Wahrscheinlichkeit, dass eine Kante ausgewählt wird nicht mehr so überschaubar bestellt.

Lemma 6.5

$$\Pr[ij \in \delta(S)] = \frac{1}{\pi} \arccos(\mathbf{y}_i \cdot \mathbf{y}_j)$$

last edit:
19.12.06

Beweis. Wir haben laut Algorithmus $ij \in \delta(S)$ genau dann, wenn die Vektoren \mathbf{y}_i und \mathbf{y}_j auf verschiedenen Seiten der von \mathbf{r} induzierten Hyperebene durch den Ursprung liegen. Man kann den allgemeinen Fall auf den zweidimensionalen Fall durch Projektion von \mathbf{r} auf die durch \mathbf{y}_i und \mathbf{y}_j aufgespannte Ebene zurückführen. Die Gerade durch den Ursprung induziert durch \mathbf{r} schneidet den Einheitskreis in genau zwei diametral gegenüberliegenden Punkten s und t . Die

Lage eines dieser Punkte ist nach Wahl von \mathbf{r} gleichverteilt auf dem Kreis. Die Vektoren \mathbf{y}_i und \mathbf{y}_j sind durch die Gerade getrennt genau dann, wenn s oder t im kürzeren Bogen zwischen \mathbf{y}_i und \mathbf{y}_j liegen. Mit der Definition des Skalarprodukts $\mathbf{y}_i \cdot \mathbf{y}_j = \|\mathbf{y}_i\| \cdot \|\mathbf{y}_j\| \cdot \cos \angle(\mathbf{y}_i, \mathbf{y}_j)$ ist die Wahrscheinlichkeit, mit der dies passiert

$$\frac{\arccos(\mathbf{y}_i \cdot \mathbf{y}_j)}{2\pi} + \frac{\arccos(\mathbf{y}_i \cdot \mathbf{y}_j)}{2\pi} = \frac{\arccos(\mathbf{y}_i \cdot \mathbf{y}_j)}{\pi} .$$

□

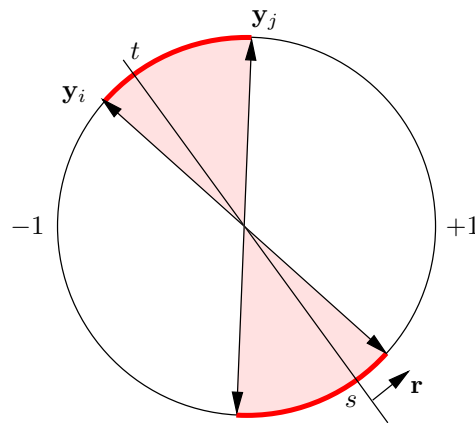


Abbildung 6.2: Bogenlänge zwischen zwei Vektoren auf dem Einheitskreis

Satz 6.6 *Der erwartete Wert eines maximalen Schnittes aus randomisiertem Runden der Lösung des Vektorprogramms (6.4) ist wenigstens $0,878 \cdot OPT$.*

Beweis. Der Beweis ist völlig analog zum Beweis von Satz 6.2. Definiere für alle $ij \in E$ eine Zufallsvariable X_{ij} mit

$$X_{ij} = \begin{cases} 1 & \text{falls } ij \in \delta(S) \\ 0 & \text{sonst} \end{cases}$$

Es sei $W = \sum_{i < j} w_{ij} X_{ij}$ der Wert des Schnittes aus dem randomisierten Runden der Lösung des Vektorprogramms 6.4. Für dessen Erwartungswert gilt

$$\begin{aligned} \mathbf{E}[W] &= \mathbf{E}\left[\sum_{i < j} w_{ij} X_{ij}\right] \\ &= \sum_{i < j} w_{ij} \cdot \Pr[ij \in \delta(S)] \\ &\stackrel{\text{Lemma 6.5}}{=} \sum_{i < j} w_{ij} \frac{1}{\pi} \arccos(\mathbf{y}_i \cdot \mathbf{y}_j) \\ &\stackrel{(6.5)}{\geq} 0,878 \cdot \frac{1}{2} \sum_{i < j} w_{ij} (1 - \mathbf{y}_i \cdot \mathbf{y}_j) \\ &= 0,878 \cdot UB \\ &\geq 0,878 \cdot OPT , \end{aligned}$$

wobei für die erste Ungleichung herangezogen wurde, dass

$$\min_{-1 \leq x \leq 1} \frac{\frac{1}{\pi} \arccos(x)}{\frac{1}{2}(1-x)} \geq 0,878 \quad . \quad (6.5)$$

gilt. Man zeigt dies etwa mittels Kurvendiskussion, s. Abbildung 6.3. □

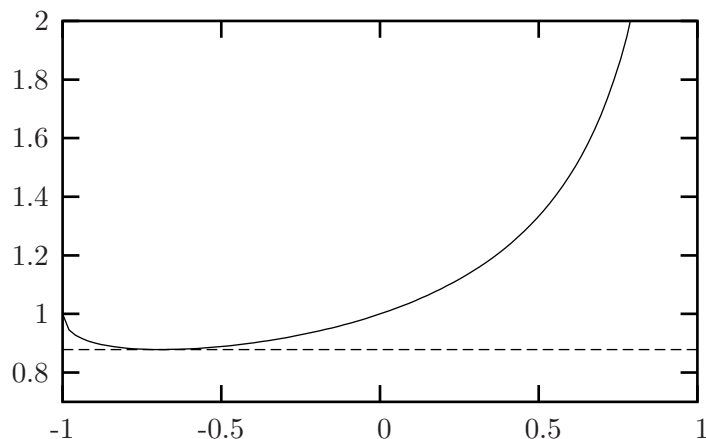


Abbildung 6.3: Verlauf der Funktion $\frac{1}{\pi} \arccos(x) / \frac{1}{2}(1-x)$ im Intervall $[-1,1]$

Bemerkung. Den Vektor \mathbf{r} wählt man komponentenweise aus der Normalverteilung mit Erwartungswert 0 und Standardabweichung 1 und anschließendes Normieren. Man kann zeigen, dass der so erhaltene Vektor gleichverteilt auf der Einheitssphäre ist.

Bemerkung. Man kann für den Kreis mit fünf Knoten nachweisen, dass die Ganzzahligkeitslücke des Vektorprogramms (6.4) $32/(25 + 5\sqrt{5}) \approx 0,88445$ ist. Das heißt, mit der semidefiniten Relaxation ist keine bessere als diese Gütegarantie zu erzielen.

Der vorgestellte Algorithmus basiert explizit auf dem Lösen eines semidefiniten Programms. David Williamson lobt \$50 für die Beantwortung folgender Frage aus.

Offenes Problem 6.7 *Gibt es einen kombinatorischen 0,878-Approximationsalgorithmus für MAX CUT?*

Bemerkung. Eine im Jahr 2004 veröffentlichte Arbeit [28] legt nahe, dass der Approximationsfaktor des Goemans-Williamson Algorithmus bestmöglich ist. Dass ausgerechnet *dieser* Faktor eine obere Schranke für die Approximierbarkeit von MAX CUT sein sollte erklären die Autoren damit, dass der Algorithmus eine spezifische geometrische Natur des Problems ausnutzt.

Kapitel 7

Facility Location und k -Median

last edit:
13.2.07

Standortplanung ist ein klassisches Thema des Operations Research. Aus Sicht der ganzzahligen Optimierung sind diese Probleme interessant, weil sie ganz einfache Beispiele für Modellierungen logischer Bedingungen mit *Indikatorvariablen* sind. In Hinblick auf Approximationsalgorithmen sind viele Varianten Gegenstand aktueller Forschung; obere und untere Schranken liegen zum Teil noch deutlich auseinander. Wir betrachten im folgenden Abschnitt die einfachste Form des FACILITY LOCATION.

7.1 Metrisches unkapazitiertes Facility Location

Gegeben ist ein vollständiger bipartiter Graph $G = (C \cup F, C \times F)$, dessen Knoten *Facilities* (oder Standorte) $i \in F$ und Städte (oder Kunden) $j \in C$ repräsentieren. Das Eröffnen eines Standorts i verursacht Kosten f_i ; das Verbinden eines Kunden j mit einem offenen Standort i kostet c_{ij} . Gesucht ist eine Teilmenge $I \subseteq F$ zu öffnender Standorte und eine Zuordnung $\phi : C \rightarrow I$ jedes Kunden zu genau einem offenen Standort. Dabei sind die Gesamtkosten, d.h. die Öffnungs- und Verbindungskosten zu minimieren.

Für beliebige Kosten lässt sich SET COVER modellieren, und damit ist keine Approximationsgüte besser als logarithmisch zu erwarten. Gilt hingegen für die Kosten c_{ij} die Dreiecksungleichung, kann man konstante Approximationsfaktoren finden. Wir entwickeln hier einen 3-Approximationsalgorithmus auf Basis des Primal-dualen Schemas. Im Unterschied zu unseren früheren Algorithmen werden wir dieses Mal die primalen Bedingungen vom komplementären Schlupf relaxieren. Um den Algorithmus auch im nächsten Abschnitt für das k -MEDIAN Problem anwenden zu können fordern wir mehr als hier nötig wäre: Die konstruierte duale Lösung „bezahlt“ die Öffnungskosten voll und wenigstens ein Drittel der Verbindungskosten.

Der zurzeit beste bekannte Approximationsalgorithmus erzielt einen Faktor von 1,52 [31]; die beste untere Schranke ist 1,463 [19]; in diesem Fall ist die Lücke damit „fast“ geschlossen. Der hier vorgestellte Algorithmus ist von Jain und Vazirani [24]. FACILITY LOCATION lässt sich als ganzzahliges Programm formulieren, dessen LP Relaxation wie folgt aussieht:

$$\begin{aligned}
\min \quad & \sum_{i \in F} \sum_{j \in C} c_{ij} x_{ij} + \sum_{i \in F} f_i y_i \\
& \sum_{i \in F} x_{ij} \geq 1 && j \in C \\
& x_{ij} \leq y_i && j \in C, i \in F \\
& x_{ij}, y_i \geq 0 && j \in C, i \in F
\end{aligned} \tag{7.1}$$

Das hierzu duale lineare Programm ist (es dualisiert sich leichter, wenn man $x_{ij} \leq y_i$ als $y_i - x_{ij} \geq 0$ liest)

$$\begin{aligned}
\max \quad & \sum_{j \in C} \alpha_j \\
& \alpha_j - \beta_{ij} \leq c_{ij} && j \in C, i \in F \\
& \sum_{j \in C} \beta_{ij} \leq f_i && i \in F \\
& \alpha_j, \beta_{ij} \geq 0 && j \in C, i \in F
\end{aligned} \tag{7.2}$$

Der Algorithmus arbeitet in zwei Phasen; zuerst bestimmen wir eine dual zulässige Lösung und eröffnen *Facilities* nur „versuchsweise“, dann bestimmen wir die tatsächlich zu öffnenden und ordnen ihnen Städte zu. Somit entsteht in der zweiten Phase eine primal zulässige, ganzzahlige Lösung. Wir lassen uns von den Bedingungen des Komplementären Schlupfs leiten:

$$x_{ij} > 0 \text{ (d.h. } x_{ij} = 1) \Rightarrow \alpha_j = c_{ij} + \beta_{ij} \tag{7.3}$$

Falls $\phi(j) = i$, so interpretieren wir α_j als Preis, den j zu zahlen bereit ist. Hiervon entfallen c_{ij} auf Verbindungskosten und β_{ij} auf Beitrag zu den Öffnungskosten für *Facility* i .

$$\beta_{ij} > 0 \Rightarrow y_i = x_{ij} \tag{7.4}$$

Falls i offen ist, aber $\phi(j) \neq i$, dann ist $y_i \neq x_{ij}$ und somit muss $\beta_{ij} = 0$ gelten. Dies bedeutet, dass nur zu i zugeordnete Städte auch zu den Öffnungskosten von i beitragen dürfen.

$$y_i > 0 \text{ (d.h. } y_i = 1) \Rightarrow \sum_{j: \phi(j)=i} b_{ij} = f_i \tag{7.5}$$

Für die Öffnungskosten einer *Facility* wird voll bezahlt. Die vierte Bedingung ist nicht interessant. Wir werden wieder Synchronisierung der Dualvariablen anwenden und man führt in diesem Zusammenhang oft den Begriff der Zeit ein: Dualvariablen wachsen alle mit derselben Rate, eine Einheit pro Zeiteinheit. Hier heben wir die α_j Variablen an; die β_{ij} reagieren lediglich so, dass die duale Zulässigkeit (bzw. der Komplementäre Schlupf) gewahrt bleibt.

Am Ende von Phase 1 (Algorithmus 13a) tragen Städte eventuell bei mehreren *Facilities* zu den Öffnungskosten bei, was wir wegen (7.4) nicht wollen. Wir nennen *Facilities* i und i' *abhängig*, wenn mit beiden eine Stadt j Kanten verbunden ist und $\beta_{ij}, \beta_{i'j} > 0$. Wir bestimmen

Algorithmus 13a Facility Location: Phase 1, versuchsweises Öffnen

Initialisiere alle Städte als *unverbunden*; setze $\alpha_j = 0$, $\beta_{ij} = 0$

while noch Städte *unverbunden* **do**

- Erhöhe α_j gleichzeitig und gleichmäßig für alle *unverbundenen* Städte $j \in C$
- falls $\alpha_j = c_{ij}$ für eine Kante ij , deklariere ij als *bezahlt* („ j hat i erreicht“)
- erhöhe β_{ij} für bezahlte Kanten ij mit gleicher Rate wie α_j
- falls $\sum_{j \in C} \beta_{ij} = f_i$, so ist i *bezahlt*; öffne i *versuchsweise*
alle *unverbundenen* Städte, die i erreicht haben, werden als *verbunden* deklariert
- erreicht eine Stadt j eine versuchsweise offene *Facility* i , wird j *verbunden*
(in diesem Fall ist $\beta_{ij} = 0$)

end while

Falls mehrere Ereignisse gleichzeitig eintreten, arbeite sie in beliebiger Reihenfolge ab.

Algorithmus 13b Facility Location: Phase 2, Zuordnung der Städte

// Vor Phase 2 wird eine inklusionsweise maximale unabhängige Menge $I \subseteq F$ geöffnet

for jede Stadt $j \in C$ **do**

- enthält I eine *Facility* i , zu deren Öffnungskosten j echt beigetragen hat ($\beta_{ij} > 0$)
setze $\phi(j) = i$ und nenne j *direkt verbunden*
- **andernfalls**: enthält I eine *Facility* i' , zu der j in Phase 1 verbunden wurde ($\beta_{ij} = 0$)
setze $\phi(j) = i'$ und nenne j *direkt verbunden*
- **andernfalls** ist $i' \notin I$, d.h. keine *Facility* i' , zu der j in Phase 1 verbunden wurde ist
offen; dann wähle eine zu i' abhängige *Facility* $i'' \in I$
(die existiert, sonst hätte i' unabhängig gewählt werden können)
setze $\phi(j) = i''$ und nenne j *indirekt verbunden*

end for

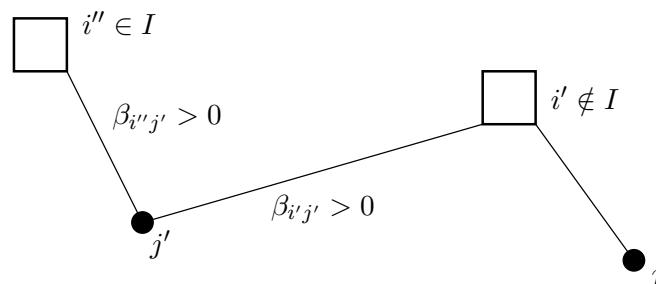


Abbildung 7.1: In Phase 1 wurde j mit i' verbunden, aber $i' \notin I$, d.h. i' wurde nicht geöffnet. Dann existiert eine zu i' abhängige *Facility* $i'' \in I$, mit der j *indirekt verbunden* wird.

dann unter den versuchsweise offenen *Facilities* eine inklusionsweise maximale unabhängige Menge I und öffnen alle $i \in I$. Insbesondere wird für jede Stadt j höchstens eine *Facility* i mit $\beta_{ij} > 0$ wirklich geöffnet. Wir setzen $y_i = 1 \iff i \in I$ und $x_{ij} = 1 \iff \phi(j) = i$.

Satz 7.1 Die in Algorithmus 13 konstruierten primalen und dualen Lösungen erfüllen

$$\sum_{i \in F} \sum_{j \in C} c_{ij} x_{ij} + 3 \sum_{i \in F} f_i y_i \leq 3 \sum_{j \in C} \alpha_j . \quad (7.6)$$

Beweis. Wir betrachten Öffnungs- und Verbindungskosten separat. Zunächst gilt für $i \in I$, dass

$$\sum_{j: \phi(j)=i} \beta_{ij} = f_i ,$$

denn $\sum_j \beta_{ij} = f_i$, d.h. i ist voll bezahlt, nur direkt verbundene Städte tragen zu Öffnungskosten bei, und alle Städte j mit $\beta_{ij} > 0$ sind auch direkt mit i verbunden. Aufsummiert über alle *Facilities* ergibt sich

$$\sum_{i \in F} \sum_{j \in C} \beta_{ij} = \sum_{i \in I} f_i = \sum_{i \in F} f_i y_i . \quad (7.7)$$

Für jede direkt verbundene Stadt j sind die Verbindungskosten $c_{\phi(j)j} = \alpha_j - \beta_{\phi(j)j}$, also insbesondere $c_{\phi(j)j} \leq 3(\alpha_j - \beta_{\phi(j)j})$. Wir zeigen schließlich, dass für eine indirekt verbundene Stadt j gilt, dass $c_{\phi(j)j} \leq 3\alpha_j$ (beachte $\beta_{\phi(j)j} = 0$), so dass Summation über alle Städte zusammen mit (7.7) die Behauptung (7.6) ergibt.

Sei $i' \notin I$ eine *Facility*, mit der j in Phase 1 verbunden wurde und $i'' \in I$ die *Facility*, mit der j in Phase 2 indirekt verbunden wurde. Weil i' und i'' nach Wahl des Algorithmus abhängig sind, gibt es eine Stadt j' , so dass

$$\beta_{i'j'} > 0 \quad \text{und} \quad \beta_{i''j'} > 0 , \quad (7.8)$$

s. Abbildung 7.1. Aufgrund der Dreieckungleichung gilt $c_{i''j} \leq c_{i'j} + c_{i'j'} + c_{i''j'}$. Wir zeigen, dass keiner der Summanden auf der rechten Seite größer ist als α_j , d.h. $c_{i''j} \leq 3\alpha_j$. Zunächst hat j die *Facility* i' erreicht, daher ist $\alpha_j \geq c_{i'j}$. Weiterhin hat j' sowohl i' als auch i'' erreicht, also gilt

$$\alpha_{j'} \geq c_{i'j'} \quad \text{und} \quad \alpha_{j'} \geq c_{i''j'} . \quad (7.9)$$

Wegen (7.8) hat j' beide *Facilities* erreicht, bevor diese versuchsweise geöffnet wurden. Nun konnte j in Phase 1 erst mit i' verbunden werden, nachdem i' versuchsweise geöffnet wurde; zusammen ist also $\alpha_j \geq \alpha_{j'}$, so dass wegen (7.9) wie benötigt $\alpha_j \geq c_{i'j'}$ und $\alpha_j \geq c_{i''j'}$. \square

Übung. Gib ein *tight example* an, das zeigt, dass die Analyse des Algorithmus scharf ist.

Übung. Zeige, dass sich jede SET COVER Instanz als Instanz des *nicht-metrischen FACILITY LOCATION* darstellen lässt, und insbesondere nicht konstant approximierbar ist.

Übung. Gib eine Implementation des Approximationsalgorithmus' an. Wie ist die Laufzeit?

Kapitel 8

Iteriertes Runden

last edit:
13.2.07

Beim Runden einer fraktionalen Lösung (eines linearen oder semidefiniten Programms) haben wir *alle* fraktionalen Variablen oberhalb eines Schwellwertes oder mit einer gewissen Wahrscheinlichkeit zur nächstgrößeren ganzen Zahl aufgerundet. In diesem Kapitel lernen wir eine allgemeiner anwendbare Spielart kennen, bei der nur jeweils *eine* Variable mit jeweils größtem Wert aufgerundet wird. Dann löst man das so modifizierte Programm neu und iteriert; daher der Name dieser von Jain [23] eingeführten Technik: *Iteriertes Runden*.

8.1 Survivable Network Design

In diesem Kapitel werden bisher betrachtete NETZWERK DESIGN Probleme verallgemeinert zum sogenannten SURVIVABLE NETWORK DESIGN Problem (auch: VERALLGEMEINERTES STEINER NETZWERK Problem). Die Motivation ist, kostengünstig ein „robustes“ Netzwerk zu installieren, dessen Funktionalität, etwa das Bedienen des Kommunikationsbedarfs zwischen einzelnen Teilnehmern, auch bei Ausfall einiger Leitungen nicht gefährdet ist. Gegeben sind

- ein ungerichteter Graph $G = (V, E)$
- eine Kostenfunktion $c : E \rightarrow \mathbb{Q}_+$ auf den Kanten
- ein *Zusammenhangsbedarf* $r : V \times V \rightarrow \mathbb{Z}_+$ zwischen je zwei Knoten und
- eine obere Schranke $u : E \rightarrow \mathbb{Z}_+ \cup \{\infty\}$ dafür, wie häufig eine Kante $e \in E$ in das zu erstellende Netzwerk aufgenommen werden darf. Wir sagen auch: Von Kante $e \in E$ dürfen höchstens u_e *Kopien* existieren.

Gesucht ist

- ein kostenminimaler Multigraph mit Knotenmenge V , der $r(u, v)$ kantendisjunkte Wege für jedes Paar $u, v \in V$ enthält. Anders gesagt: Jeder u - v -Schnitt muss mindestens $r(u, v)$ Kanten(-kopien) enthalten. Von einer Kante $e \in E$ dürfen höchstens u_e Kopien angelegt werden; jede Kopie der Kante $e \in E$ kostet c_e .

Alle in Abschnitt 4.2 genannten Probleme sind Spezialfälle dieser Aufgabenstellung und wir haben gesehen, dass das Primal-duale Schema fast in jedem Fall eine 2-Approximation

liefert. Für den allgemeinsten hier zu betrachtenden Fall kann man jedoch nur einen Approximationsfaktor von $2 \cdot H_k$ nachweisen, wobei $k = \max_{u,v \in V} r(u,v)$. Iteriertes Runden wird uns auch hier auf einen 2-Approximationsalgorithmus führen.

Ein ganzzahliges Programm

Für jede Knotenteilmenge $S \subseteq V$ schreiben wir den *Zusammenhangsbedarf des Schnitts* $\delta(S)$ als

$$f(S) = \max_{u \in S, v \notin S} r(u,v) . \quad (8.1)$$

Damit können wir als Verallgemeinerung des ganzzahligen Programms (4.13) das SURVIVABLE NETWORK DESIGN Problem formulieren als

$$\begin{aligned} OPT = \min \quad & \sum_{e \in E} c_e x_e \\ & \sum_{e \in \delta(S)} x_e \geq f(S) \quad S \subseteq V \\ & x_e \in \{0, 1, \dots, u_e\} \quad e \in E , \end{aligned} \quad (8.2)$$

mit der LP-Relaxation

$$\begin{aligned} LB = \min \quad & \sum_{e \in E} c_e x_e \\ & \sum_{e \in \delta(S)} x_e \geq f(S) \quad S \subseteq V \\ & 0 \leq x_e \leq u_e \quad e \in E . \end{aligned} \quad (8.3)$$

Beim Aufrunden einer fraktionalen Lösung der linearen Relaxation des SET COVER Problems wussten wir, dass positive Variablen einen gewissen Schwellwert überschreiten (Satz 3.6). Bei VERTEX COVER haben wir mit Satz 3.13 sogar bewiesen, dass eine fraktionale Lösung immer halbzahlige ist. Diese günstige Eigenschaft können wir hier leider nicht erwarten.

Lemma 8.1 *Für den Petersen-Graph mit $r(u,v) = 1$ zwischen je zwei Knoten u und v hat die LP-Relaxation (8.3) keine halbzahlige Optimallösung.*

Beweis. Abbildung 8.1 zeigt den Petersen-Graph. Die verschiedenen Linienstärken sind hier uninteressant. Der Petersen-Graph ist 3-zusammenhängend, daher ist $x_e = 1/3$ für alle $e \in E$ zulässig: Zwischen je zwei Knoten gibt es drei Wege mit Wert je $1/3$. Dies ist optimal, da $LB \geq 5$, weil die Summe der Kantenvariablen an jedem der zehn Knoten wenigstens 1 sein muss, um den Zusammenhangsbedarf zu sichern. Die Kantenvariablen in einer halbzahligen Lösung dürfen die Werte 0, $1/2$ oder 1 annehmen. Jede Lösung mit $x_e = 1$ für eine Kante $e = (u,v)$ hat Kosten größer als 5, denn der Zusammenhangsbedarf von u und v zu allen anderen Knoten muss über weitere Kanten gedeckt werden. Daher muss eine optimale halbzahlige Lösung zwischen je zwei Knoten zwei kantendisjunkte Wege mit Wert je $1/2$ enthalten, mit anderen Worten die Kanten eines Hamiltonschen Kreises müssen Wert je $1/2$ annehmen (für 10 Kanten mit Wert je $1/2$ ergäbe dies genau den Wert 5). Da der Petersen-Graph nicht Hamiltonsch ist, kann es eine solche Lösung nicht geben. \square

Das Approximationsresultat dieses Kapitels baut auf der Kenntnis der Struktur der Basislösungen des linearen Programms (8.3). Ein Beispiel für eine optimale Basislösung ist in Abbildung 8.1 gegeben. Obwohl die dargestellte Lösung nicht halbzahlig ist, haben *einige* Kanten den Wert $1/2$. Dies ist kein Zufall, sondern beweisbare Eigenschaft der LP-Relaxation (8.3).

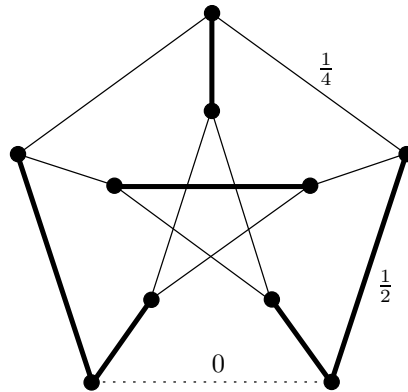


Abbildung 8.1: Eine optimale Basislösung der LP-Relaxation (8.3) für den Petersen-Graph: Dick gezeichnete Kanten haben Wert $1/2$, dünn gezeichnete Kanten haben Wert $1/4$ und die gepunktet gezeichnete Kante hat Wert 0 .

Wir verwenden die untere Schranke aus dem ursprünglichen linearen Programm (8.3), dieses wird aber im Laufe der Iterationen durch Aufrunden (und Fixieren) einzelner Variablen verändert. Die am Ende konstruierte Lösung bleibt allerdings zulässig für das originale LP.

Submodulare und schwach supermodulare Funktionen

Beschäftigen wir uns nun mit den Eigenschaften der von uns in (8.1) definierten Funktion $f : 2^V \rightarrow \mathbb{Z}_+$ des Zusammenhangsbedarfs.

Definition 8.2 Eine Funktion $f : 2^V \rightarrow \mathbb{Z}_+$ heißt stark submodular falls $f(V) = 0$ und für je zwei Mengen $A, B \subseteq V$ gilt

1. $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ und
2. $f(A) + f(B) \geq f(A \setminus B) + f(B \setminus A)$

Für Submodularität wird nur die erste Eigenschaft gefordert; wir benutzen hier die stärkere Definition. Ein klassischer Vertreter für eine (sogar stark) submodulare Funktion in der kombinatorischen Optimierung ist die Schnittfunktion.

Lemma 8.3 Die auf einem ungerichteten Graph $G = (V, E)$ definierte Funktion $\delta : 2^V \rightarrow \mathbb{Z}_+$ mit $S \mapsto |\delta(S)|$, $S \subseteq V$ ist stark submodular.

Beweis. Seien $A, B \subseteq V$ gegeben. Gilt $A \cap B = \emptyset$ oder gilt $A \subseteq B$ oder $B \subseteq A$, so sind die beiden Bedingungen in Definition 8.2 mit Gleichheit erfüllt.

Sei dies nicht der Fall, d.h. A und B kreuzen sich, vgl. Abbildung 8.2. Eine Kante e_1 mit einem Endknoten in $A \setminus B$ und dem anderen in $B \setminus A$ trägt zu $\delta(A)$ und $\delta(B)$ bei, nicht aber

zu $\delta(A \cup B)$ oder $\delta(A \cap B)$. Eine Kante e_2 mit einem Endknoten in $A \cap B$ und dem anderen nicht in $A \cup B$ trägt zu $\delta(A)$ und $\delta(B)$ bei, nicht aber zu $\delta(A \setminus B)$ oder $\delta(B \setminus A)$. Jeweils alle anderen Kanten tragen gleichzeitig zu beiden Seiten der beiden Bedingungen bei. \square

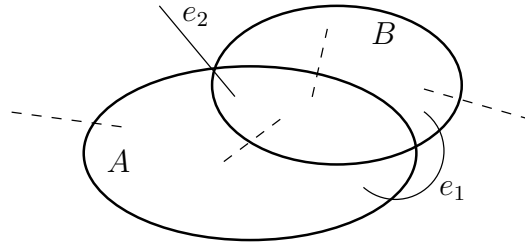


Abbildung 8.2: Sich kreuzende Mengen A und B im Beweis von Lemma 8.3

Definition 8.4 Eine Funktion $f : 2^V \rightarrow \mathbb{Z}_+$ heißt schwach supermodular falls $f(V) = 0$ und für je zwei Mengen $A, B \subseteq V$ wenigstens eine der folgenden Eigenschaften gilt

1. $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$ oder
2. $f(A) + f(B) \leq f(A \setminus B) + f(A \setminus B)$

Lemma 8.5 Die Funktion $f : 2^V \rightarrow \mathbb{Z}_+$ mit $S \mapsto f(S) = \max_{u \in S, v \notin S} r(u, v)$ ist schwach supermodular.

Beweis. Der Beweis läuft ähnlich dem des vorangegangenen Lemmas, nur dass jetzt Zusammenhangsbedarfe der Mengen $A, B \subseteq V$ betrachtet werden, s. Abbildung 8.3. Gilt $A \cap B = \emptyset$, so trägt jeder Bedarf r_1 , der zu $f(A)$ und/oder $f(B)$ beiträgt sowohl zu $f(A \setminus B)$ als auch zu $f(B \setminus A)$ in gleichem Maße bei. Die zweite Bedingung aus Definition 8.4 gilt.

Sei $A \subseteq B$. Ein Bedarf r_2 von A nach B trägt zu $f(A)$ und zu $f(B \cap A)$ bei. Ein Bedarf r_3 aus B heraus trägt zu $f(B)$ und zu $f(A \cup B)$ bei. Ein Bedarf r_4 aus A hinaus aus $A \cup B$ trägt zu $f(A)$, $f(B)$, $f(A \cup B)$ und $f(A \cap B)$ bei. Analog betrachtet man den Fall $B \subseteq A$. Die erste Bedingung der Definition gilt.

Kreuzen sich A und B , so unterscheiden wir, ob der Bedarf r_5 aus $A \setminus B$ hinaus nach $B \setminus A$ oder der Bedarf r_6 aus $A \cap B$ hinaus aus $A \cup B$ größer ist. Im ersten Fall ist $f(A) = f(B) = f(A \setminus B) = f(B \setminus A) = r_5$ und die zweite Bedingung der Definition gilt. Im zweiten Fall ist $f(A) = f(B) = f(A \cup B) = f(B \cap A) = r_5$ und die erste Bedingung gilt. Alle anderen Bedarfe, sollten sie größer als $\max\{r_5, r_6\}$ sein, tragen gleichermaßen zu beiden Seiten beider Bedingungen bei. \square

In einer Iteration des Algorithmus wird das Aufrunden einzelner Variablen x_e so realisiert, dass der entsprechende Zusammenhangsbedarf aller Knotenmengen $S \subseteq V$ mit $e \in \delta(S)$ vermindert wird. Dies verändert (genauer: senkt) in jeder Iteration den (verbleibenden) Zusammenhangsbedarf, wir sprechen von *Residual-Zusammenhangsbedarf*.

Lemma 8.6 Sei $f : 2^V \rightarrow \mathbb{Z}_+$ schwach supermodulare Funktion und H eine Multimenge von Kanten. Dann ist auch die Funktion des Residual-Zusammenhangsbedarfs $f' : 2^V \rightarrow \mathbb{Z}_+$ mit $S \mapsto f'(S) = f(S) - |\delta_H(S)|$, $S \subseteq V$ schwach supermodular.

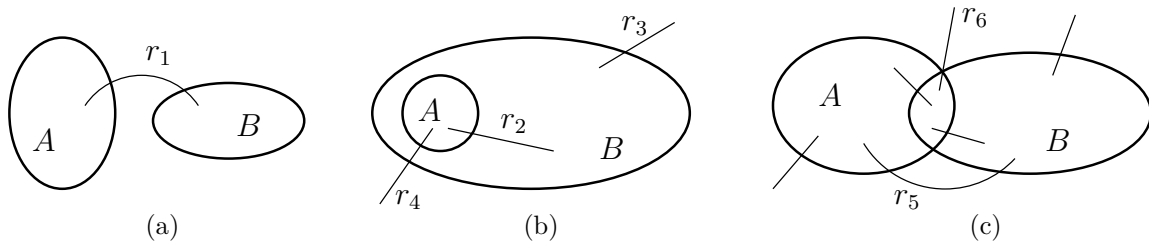


Abbildung 8.3: Zum Beweis des Lemmas 8.5

Beweis. Sei $f(A) + f(B) \leq f(A \cup B) + f(A \cap B)$. Wegen der starken Submodularität der Schnittfunktion (Lemma 8.3) gilt $|\delta_H(A)| + |\delta_H(B)| \geq |\delta_H(A \cup B)| + |\delta_H(A \cap B)|$. Subtraktion der Schnittfunktion vermindert die rechte Seite also nicht stärker als die linke, und daher gilt $f'(A) + f'(B) \leq f'(A \cup B) + f'(A \cap B)$. Ebenso zeigt man die zweite Ungleichung. \square

Damit können wir einen starken Satz der polyedrischen Kombinatorik formulieren.

Satz 8.7 Sei $f : 2^V \rightarrow \mathbb{Z}_+$ eine schwach supermodulare Funktion. Für jede Basislösung \mathbf{x} des linearen Programms (8.3) gibt es eine Variable x_e mit $x_e \geq \frac{1}{2}$.

Der Beweis dieses Satzes benutzt keine Mittel, die wir nicht zur Verfügung stellen könnten, beansprucht allerdings bereits in der Originalarbeit [23] mehr als sechs Seiten, so dass wir ihn hier leider aussparen. Wir geben nun den Algorithmus formal an.

Algorithmus 14 Iteriertes Runden für SURVIVABLE NETWORK DESIGN

Initialisierung: $H = \emptyset$, $f' = f$

while $f' \neq \mathbf{0}$ **do**

Bestimme optimale Basislösung \mathbf{x} zum LP (8.3) mit Residual-Zusammenhangsbedarf f'

for jede Kante $e \in E$ mit $x_e \geq 1/2$ **do**

Nimm $\lceil x_e \rceil$ Kopien der Kante e in H auf

Setze $u_e = u_e - \lceil x_e \rceil$

end for

Setze $f'(S) = f(S) - |\delta_H(S)|$ für alle $S \subseteq V$

end while

Gib H aus.

Laufzeit

In Bezug auf die polynomiale Laufzeit des vorgeschlagenen Algorithmus müssen wir zwei Schritte besonders untersuchen. Zum einen muss ein lineares Programm mit exponentiell vielen Restriktionen optimal gelöst werden. Die Ellipsoid-Methode ist hierzu in der Lage, wenn wir ein polynomiales Separationsorakel angeben können, das zu gegebener Lösung \mathbf{x} eine verletzte Restriktion identifiziert oder feststellt, dass \mathbf{x} zulässig ist. Der zweite Punkt ist das Update der exponentiell vielen Residual-Zusammenhangsbedarfe.

In der ersten Iteration können wir eine verletzte Restriktion mit einem Maximaler-Fluss-Algorithmus bestimmen. Zu gegebener Lösung \mathbf{x} belegen wir jede Kante $e \in E$ des Graph G

mit Kapazität x_e . Dann überprüfen wir für jedes Knotenpaar (u, v) , ob ein maximaler Fluss von u nach v wenigstens den Wert $r(u, v)$ erreicht. Falls nicht, erhalten wir einen verletzten Schnitt, d.h. eine verletzte Restriktion mit

$$\delta_{\mathbf{x}}(S) := \sum_{e \in \delta(S)} x_e < f(S) .$$

Sei nun f' Residual-Zusammenhangsbedarf in einer späteren Iteration. Zu gegebener Lösung \mathbf{x}' des modifizierten linearen Programms definieren wir \mathbf{x} wie folgt. Für jede Kante $e \in E$ setze $x_e = x'_e + e_H$, wobei e_H die Anzahl der Kopien der Kante e in H bezeichne. Das folgende Lemma zeigt, dass unser polynomiales Separationsorakel für die originale Funktion f zu einem polynomialen Separationsorakel für die Residual-Zusammenhangsbedarfsfunktion f' führt. Wir werden gleichzeitig sehen, dass damit gar keine Notwendigkeit mehr besteht, einen tatsächlichen Update-Schritt der Funktion f (bzw. f') durchzuführen.

Lemma 8.8 *Ein Schnitt $\delta(S)$ ist von einer Lösung \mathbf{x}' zu Residual-Zusammenhangsbedarf f' verletzt genau dann, wenn er von Lösung \mathbf{x} zu Zusammenhangsbedarf f verletzt ist.*

Beweis. Zunächst gilt $\delta_{\mathbf{x}}(S) = \delta_{\mathbf{x}'}(S) + |\delta_H(S)|$ nach unserer Definition von \mathbf{x}' . Weiterhin haben wir $f'(S) = f(S) - |\delta_H(S)|$ definiert. Zusammen gibt das

$$\begin{aligned} \delta_{\mathbf{x}}(S) &= \delta_{\mathbf{x}'}(S) + f(S) - f'(S) \\ \iff \delta_{\mathbf{x}}(S) - f(S) &= \delta_{\mathbf{x}'}(S) - f'(S) , \end{aligned}$$

d.h. $\delta_{\mathbf{x}}(S) \geq f(S) \iff \delta_{\mathbf{x}'}(S) \geq f'(S)$, was zu beweisen war. \square

Positiv formuliert lautet die Aussage dieses Lemmas, dass eine Lösung \mathbf{x}' für das lineare Programm (8.3) mit rechter Seite f' zulässig ist, genau dann, wenn die Lösung \mathbf{x} mit rechter Seite f zulässig ist. Unter der Voraussetzung, dass die polyedrische Aussage des Satz 8.7 gilt, können wir nun die behauptete Approximationsgarantie beweisen.

Satz 8.9 *Iteriertes Runden ist ein 2-Approximationsalgorithmus für das SURVIVABLE NETWORK DESIGN Problem.*

Beweis. Sei f eine schwach supermodulare Zusammenhangsbedarfsfunktion. Wir zeigen, dass die Kosten der konstruierten ganzzahligen Lösung H höchstens doppelt so hoch sind wie die Kosten LB der optimalen fraktionalen Lösung zu (8.3). Weil $LB \leq OPT$ folgt die Behauptung. Der Beweis läuft induktiv über die Anzahl der Iterationen.

Terminiert der Algorithmus nach einer Iteration, so enthält H nur Kanten e mit $x_e \geq 1/2$ und die Behauptung folgt sofort.

Für den Induktionsschritt sei \mathbf{x} die in der ersten Iteration bestimmte Basislösung. Aus \mathbf{x} leiten wir einen Vektor $\hat{\mathbf{x}}$ durch Nullsetzen aller Komponenten mit $x_e < 1/2$ ab. Aufgrund von Satz 8.7 ist $\hat{\mathbf{x}} \neq \mathbf{0}$. Sei H die Multimenge der in der ersten Iteration gewählten Kanten. Wir bezeichnen etwas missbräuchlich mit $c(\cdot)$ die Kosten der Multimenge, bzw. der zugehörigen Lösung. Weil H durch Aufrunden von Variablen in $\hat{\mathbf{x}}$ entstanden ist, gilt $c(H) \leq 2 \cdot c(\hat{\mathbf{x}})$.

Sei f' der Residual-Zusammenhangsbedarf nach der ersten Iteration und H' die Multimenge der Kanten, die nach der ersten Iteration aufgenommen wurden, um f' zu erfüllen.

Die wichtige Beobachtung ist, dass $\mathbf{x} - \hat{\mathbf{x}}$ zulässig ist für (8.3) mit rechter Seite f' . Für die Kosten von H' gilt daher nach Induktionsvoraussetzung $c(H') \leq 2 \cdot c(\mathbf{x} - \hat{\mathbf{x}})$. Weiterhin gilt dass die Multimenge $H \cup H'$ den Zusammenhangsbedarf f erfüllt. Zusammen ergibt sich

$$\begin{aligned} c(H \cup H') &\leq c(H) + c(H') \leq 2 \cdot c(\hat{\mathbf{x}}) + 2 \cdot c(\mathbf{x} - \hat{\mathbf{x}}) \leq 2 \cdot c(\mathbf{x}) \\ &= 2 \cdot LB \leq 2 \cdot OPT . \end{aligned}$$

□

Korollar 8.10 Die Ganzzahligkeitslücke des linearen Programms (8.3) ist höchstens 2.

Bemerkung. Weil f in jeder Iteration des Algorithmus echt vermindert wird, wissen wir zwar, dass iteriertes Runden terminiert, nicht aber wie viele Iterationen nötig sind. Die Größe der Zahlen in f , letztlich also die Größe der Kapazitäten u_e , $e \in E$, geht in die Laufzeit des Algorithmus ein, der damit nur pseudopolynomial ist. Jain beschreibt in [23] eine stark polynomiale Variante.

Bemerkung. Man beachte, dass wir Lemma 8.8 nicht für *irgendeine* schwach supermodulare Funktion beweisen können. Damit gilt zwar die Approximationsgarantie für jede schwach supermodulare Funktion, nicht aber die polynomiale Laufzeit des iterierten Rundens.

Bemerkung. Eine Funktion $f : 2^V \rightarrow \mathbb{Z}_+$ heißt *maximal* (vgl. Definition 4.10), wenn für disjunkte $A, B \subseteq V$ gilt $f(A \cup B) \leq \max\{f(A), f(B)\}$. Man kann man leicht nachweisen, dass jede echte Funktion $f : 2^V \rightarrow \mathbb{Z}_+$ schwach supermodular ist, iteriertes Runden daher eine 2-Approximation für alle in Kapitel 4 betrachteten NETZWERK DESIGN Probleme gibt.

Übung. Gib eine „kompakte“ Formulierung als ganzzahliges Programm für das VERALLGEMEINERTE STEINER NETZWERK Problem, d.h. mit polynomial vielen Variablen und Restriktionen. Hinweis: Fasse das Problem als MULTICOMMODITY FLOW Problem mit einer Commodity für jedes Knotenpaar auf. Jeder Fluss muss mindestens so groß sein, wie der Zusammenhangsbedarf des Knotenpaares. Eine Kante kann nur soviel Fluss tragen, wie Kapazität auf ihr bereit gestellt wird.

Kapitel 9

Maximum Satisfiability

last edit:
13.2.07

Zum Ende der Vorlesung behandeln wir einen „Klassiker“, das MAXIMUM SATISFIABILITY Problem oder kurz MAX SAT. Wieder wird randomisiertes LP-Runden eine Rolle spielen und wir lernen eine Standardtechnik zum Derandomisieren kennen. Darüberhinaus wird uns dieses Problem an die jüngsten Entwicklungen in der Theorie der (Nicht-)Approximierbarkeit heranführen.

Gegeben sei eine Formel f in konjunktiver Normalform in Booleschen Variablen x_1, \dots, x_n , d.h. mit der Menge \mathcal{C} der Klauseln ist f von der Form

$$f = \bigwedge_{c \in \mathcal{C}} c .$$

Als *Literal* bezeichnet man eine Variable x_i oder ihre Negation \bar{x}_i . Jede Klausel $c \in \mathcal{C}$ ist Disjunktion von Literalen und hat ein Gewicht $w_c \geq 0$. Eine Klausel ist demnach *erfüllt*, wenn wenigstens eins ihrer Literale *wahr* ist. Gesucht ist eine Wahrheitsbelegung der Variablen so, dass das Gewicht aller erfüllten Klauseln maximal ist. Die Anzahl der Literale einer Klausel c bezeichnen wir mit $|c|$. Fordern wir $|c| \leq k$, $c \in \mathcal{C}$ für ein $k \in \mathbb{N}$, so heißt dieses eingeschränkte Problem MAX k SAT. Selbst MAX 2SAT ist \mathcal{NP} -schwer.

Lange Klauseln

Wir haben schon mehrere Male gesehen, dass ein einfacher Vorschlag auf einen guten Algorithmus führen kann.¹ Die folgende Idee hatte Johnson bereits im Jahr 1974 [25]: Für jede Variable werfen wir eine Münze.

Algorithmus 15 Randomisiertes Runden für MAX SAT

```
for  $x_i = x_1, \dots, x_n$  do  
  Setze  $x_i$  auf wahr mit der Wahrscheinlichkeit 1/2  
end for  
Gib die Wahrheitsbelegung aus
```

¹Wir lernen daraus, dass wir die möglicherweise absurd naheliegenden algorithmischen Ideen ruhig einmal ausprobieren sollten, wenn wir an ein neues Problem herangehen.

Satz 9.1 *Randomisiertes Runden ist ein randomisierter 1/2-Approximationsalgorithmus für MAX SAT.*

Beweis. Seien $W_c, c \in \mathcal{C}$, Zufallsvariablen, die jeweils den Wert des Gewichts annehmen, das die Klausel c unter der im Algorithmus gewählten Wahrheitsbelegung zur Zielfunktion beiträgt. Sei W eine Zufallsvariable, die den Wert des Gewichts aller erfüllten Klauseln annimmt, also $W = \sum_{c \in \mathcal{C}} W_c$.

Eine Klausel c ist nicht erfüllt, wenn alle ihre Literale *falsch* sind. Hat c die Länge $|c| = k$, so ist die Wahrscheinlichkeit, dass dieses Ereignis eintritt 2^{-k} und daher

$$\mathbf{E}[W_c] = (1 - 2^{-k})w_c =: \alpha_k w_c . \quad (9.1)$$

Für $k \geq 1$ ist $\alpha_k \geq 1/2$. Somit gilt für den erwarteten Zielfunktionswert

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geq \frac{1}{2} \sum_{c \in \mathcal{C}} w_c \geq \frac{1}{2} OPT .$$

Wir haben hier die triviale obere Schranke $\sum_{c \in \mathcal{C}} w_c \geq OPT$ ausgenutzt. \square

Tatsächlich lesen wir aus dem Beweis, dass Randomisiertes Runden ein 2^{-k} -Approximationsalgorithmus für MAX SAT ist, wenn $|c| \geq k, c \in \mathcal{C}$, also insbesondere erhalten wir für $k \geq 2$ eine Garantie von $3/4$. Lediglich die Klauseln mit nur einem Literal verderben diese Güte. Wir kommen später auf diese Beobachtung zurück.

Derandomisierung mit der Methode der bedingten Erwartungswerte

Wir derandomisieren nun den vorgenannten Algorithmus, d.h. wir wollen einen deterministischen Algorithmus angeben, dessen Zielfunktionswert wenigstens $\mathbf{E}[W]$ ist.

Wir beobachten zunächst die *Selbstreduzierbarkeit* genannte Eigenschaft vieler Probleme in \mathcal{NPO} , insbesondere des MAX SAT Problems. Sei f eine Formel wie oben. Fixieren wir x_1 auf entweder *wahr* oder *falsch*, so können wir in Polynomialzeit zwei Formeln f_0 und f_1 in den Variablen x_2, \dots, x_n angeben, deren alle Klauseln erfüllende Wahrheitsbelegungen genau zu den alle Klauseln erfüllenden Wahrheitsbelegungen von f mit $x_1 = \textit{wahr}$ bzw. $x_1 = \textit{falsch}$ erweitert werden können.

Beispiel. Sei $f = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$. Zu $x_1 = \textit{wahr}$ erhalten wir $f_0 = (x_2 \vee x_4)$; zu $x_1 = \textit{falsch}$ erhalten wir $f_1 = (x_2 \vee x_3)$.

Setzen wir das Fixieren der Variablen rekursiv fort, so können wir mit einem Orakel für die Entscheidungsversion von MAX SAT in Polynomialzeit eine *Lösung* für f unter der Annahme konstruieren, dass f erfüllbar ist. Entscheide dazu zuerst, ob f_0 erfüllbar ist. Falls ja, setze $x_1 = \textit{wahr}$ und finde eine Lösung zu f_0 . Ansonsten muss f_1 erfüllbar sein und wir bestimmen eine Lösung zu f_1 . In jedem Fall ist eine Lösung zu einer Formel zu finden, die eine Variable weniger enthält als die Ausgangsformel, d.h. die Rekursion hat die Tiefe n . Wir können dieses Verfahren auch als Baum repräsentieren, deren Wurzel der Formel f entspricht. Innere Knoten der Tiefe i entsprechen Formeln in den Variablen x_{i+1}, \dots, x_n ; die Variablen x_1, \dots, x_i sind also bereits auf Wahrheitswerte fixiert.

Sei T dieser Baum und a_1, \dots, a_i eine Wahrheitsbelegung der Variablen x_1, \dots, x_i . Zu jedem Knoten der Tiefe i merken wir uns $\mathbf{E}[W \mid x_1 = a_1, \dots, x_i = a_i]$, d.h. den bedingten Erwartungswert der Wahrheitsbelegung aller Variablen durch randomisiertes Runden unter der Annahme, dass die ersten i Variablen bereits eine Wahrheitsbelegung erhalten haben.

Lemma 9.2 *Die bedingte Erwartung $\mathbf{E}[W \mid x_1 = a_1, \dots, x_i = a_i]$ kann in jedem Knoten von T in Polynomialzeit berechnet werden.*

Beweis. Es sei eine Wahrheitsbelegung der Variablen x_1, \dots, x_i in einem Knoten fixiert und f_i die zugehörige Formel in den restlichen Variablen x_{i+1}, \dots, x_n . Das erwartete Gewicht aller erfüllten Klauseln von f_i bei randomisiertem Runden der Variablen x_{i+1}, \dots, x_n ist offenbar polynomial berechenbar. Hierzu addiert man das Gewicht der bereits durch die Wahrheitsbelegung der Variablen x_1, \dots, x_i in f erfüllten Klauseln. \square

Bemerkung. Nur weil die Wahrheitsbelegung der Variablen x_{i+1}, \dots, x_n von der Wahrheitsbelegung der Variablen x_1, \dots, x_i unabhängig ist, können wir das erwartete Gewicht von f_i bestimmen.

Satz 9.3 *Ein Weg von der Wurzel zu einem Blatt in T mit der Eigenschaft, dass die bedingte Erwartung in jedem Knoten entlang des Weges wenigstens $\mathbf{E}[W]$ beträgt, kann in Polynomialzeit bestimmt werden.*

Beweis. Es sei eine Wahrheitsbelegung der Variablen $x_1 = a_1, \dots, x_i = a_i$ in einem Knoten fixiert. Es ist gleich wahrscheinlich, dass x_{i+1} im Algorithmus auf *wahr* oder auf *falsch* gesetzt wird. Daher ist die bedingte Erwartung in einem Knoten gerade der Durchschnitt der bedingten Erwartungen seiner Kinder:

$$\begin{aligned} \mathbf{E}[W \mid x_1 = a_1, \dots, x_i = a_i] &= \mathbf{E}[W \mid x_1 = a_1, \dots, x_i = a_i, x_{i+1} = \text{wahr}] / 2 + \\ &\quad \mathbf{E}[W \mid x_1 = a_1, \dots, x_i = a_i, x_{i+1} = \text{falsch}] / 2 . \end{aligned}$$

Daher muss eines der Kinder eine bedingte Erwartung wenigstens so groß wie die bedingte Erwartung des Elternknoten haben. Dieses Argument induktiv von der Wurzel mit bedingter Erwartung $\mathbf{E}[W]$ ausgehend zu den Blättern hin geführt ergibt die Existenz des behaupteten Weges. Wegen Lemma 9.2 ist dieser Weg in Polynomialzeit zu finden. \square

Das deterministische Pendant des randomisierten Rundens für MAX SAT folgt aus diesem Lemma. Iterativ werden die Variablen jeweils mit demjenigen Wahrheitswert belegt, der auf den größeren bedingten Erwartungswert führt.

Kurze Klauseln

Goemans und Williamson [14] haben ein ganzzahliges Programm für das MAX SAT Problem vorgeschlagen. Für jede Klausel $c \in \mathcal{C}$ bezeichne S_c^+ die Menge der nicht negierten Literale, S_c^- die Menge der negierten. Eine binäre Variable y_i entscheidet über die Wahrheitsbelegung

der Variablen x_i : $y_i = 1$ steht für $x_i = \text{wahr}$, $y_i = 0$ bedeutet $x_i = \text{falsch}$.

$$\begin{aligned}
 OPT = \max \quad & \sum_{c \in \mathcal{C}} w_c z_c \\
 & \sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c \quad c \in \mathcal{C} \\
 & z_c \in \{0, 1\} \quad c \in \mathcal{C} \\
 & y_i \in \{0, 1\} \quad i = 1, \dots, n
 \end{aligned} \tag{9.2}$$

Die Restriktionen sichern, dass die Indikatorvariable z_c nur dann auf 1 (Klausel c erfüllt) gesetzt werden darf, wenn wenigstens eines der Literale der Klausel *wahr* ist. Die LP-Relaxation lautet

$$\begin{aligned}
 UB = \max \quad & \sum_{c \in \mathcal{C}} w_c z_c \\
 & \sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c \quad c \in \mathcal{C} \\
 & 0 \leq z_c \leq 1 \quad c \in \mathcal{C} \\
 & 0 \leq y_i \leq 1 \quad i = 1, \dots, n
 \end{aligned} \tag{9.3}$$

Algorithmus 16 Randomisiertes LP-Runden für MAX SAT

Sei $(\mathbf{y}^*, \mathbf{z}^*)$ optimale Lösung des linearen Programms (9.3)

for $x_i = x_1, \dots, x_n$ **do**

 Setze x_i auf *wahr* mit der Wahrscheinlichkeit y_i^*

end for

Gib die Wahrheitsbelegung aus

Satz 9.4 *Randomisiertes LP-Runden ist ein randomisierter $(1 - \frac{1}{e})$ -Approximationsalgorithmus für MAX SAT.*

Beweis. Seien W und W_c , $c \in \mathcal{C}$, Zufallsvariablen wie im Beweis von Satz 9.1. Wir bestimmen zunächst $\mathbf{E}[W_c]$ für eine Klausel c mit $|c| = k$ Literalen. Zur Vereinfachung der Darstellung nehmen wir ohne Einschränkung an, dass alle Literale in c nicht negiert vorkommen. Andernfalls ersetzt man \bar{x}_i durch x_i in ganz f und ersetzt y_i durch $1 - y_i$ im linearen Programm (9.3). Weiterhin benennen wir die Variablen so um, dass $c = (x_1 \vee \dots \vee x_k)$.

Die Klausel c ist erfüllt, wenn nicht alle Variablen *falsch* gesetzt sind. Die Wahrscheinlichkeit, dass dieses Ereignis eintritt, ist

$$\begin{aligned}
 1 - \prod_{i=1}^k (1 - y_i) & \geq 1 - \left(\frac{\sum_{i=1}^k (1 - y_i)}{k} \right)^k = 1 - \left(1 - \frac{\sum_{i=1}^k (y_i)}{k} \right)^k \\
 & = 1 - \left(1 - \frac{z_c^*}{k} \right)^k .
 \end{aligned}$$

Die erste Ungleichung folgt hier aus der Standardbeziehung

$$\frac{a_1 + \dots + a_k}{k} \geq \sqrt[k]{a_1 \cdot \dots \cdot a_k}$$

zwischen dem arithmetischen und geometrischen Mittel nichtnegativer Zahlen a_1, \dots, a_k . Die zweite Ungleichung gilt wegen der Restriktion $y_1 + \dots + y_k \geq z_c$ des linearen Programms (9.3).

Betrachten wir nun die Funktion

$$g(z) = 1 - \left(1 - \frac{z}{k}\right)^k .$$

Diese Funktion ist konkav mit $g(0) = 0$ und $g(1) = 1 - \left(1 - \frac{1}{k}\right)^k =: \beta_k$. Auf dem Intervall $[0, 1]$ der zulässigen Werte für z_c ist $g(z) \geq \beta_k \cdot z$, weil letzteres die Geradengleichung der Sekante unterhalb der Kurve von g ist, s. Abbildung 9.1.

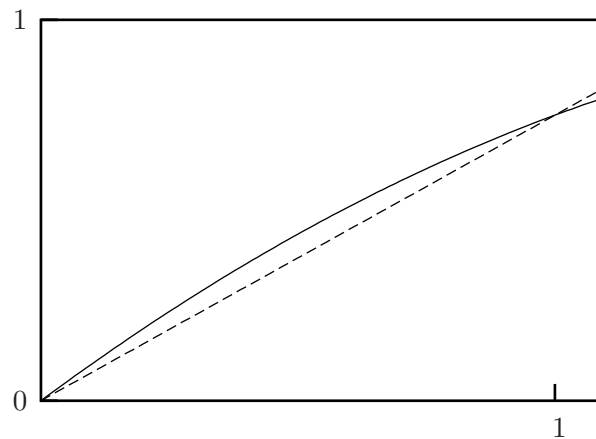


Abbildung 9.1: Graph der Funktion $g(z)$ im Beweis von Satz 9.4

Hieraus ergibt sich $\Pr[c \text{ ist erfüllt}] \geq \beta_k \cdot z_c^*$ und

$$\mathbf{E}[W_c] \geq \beta_k \cdot w_c \cdot z_c^* . \quad (9.4)$$

Haben alle Klauseln höchstens k Literale schätzen wir wie oben ab

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geq \beta_k \sum_{c \in \mathcal{C}} w_c z_c^* = \beta_k UB \geq \beta_k OPT .$$

In der ersten Ungleichung haben wir ausgenutzt, dass β_k mit wachsendem k fällt, d.h. $\beta_1 \geq \beta_2 \geq \dots \geq \beta_k$, die linke Seite also höchstens größer wird, wenn eine Klausel weniger als k Literale enthält. Schließlich gilt

$$\beta_k = 1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e}$$

und die Behauptung folgt. □

Übung. Derandomisiere LP-Runden mit der Methode der bedingten Erwartungen.

„Best of two“ Algorithmus

Der Beweis von Satz 9.4 liefert für MAX k SAT die etwas bessere Garantie von β_k für LP-Runden. Insbesondere ist das eine Garantie von $3/4$, wenn $k \leq 2$ und interessanterweise genau komplementär zu der Garantie von $3/4$ für $k \geq 2$ für den zuerst vorgestellten Algorithmus von Johnson. Dies führt auf folgenden deterministischen Algorithmus.

Algorithmus 17 „Best of two“ für MAX SAT

Bestimme mit derandomisiertem Runden eine Wahrheitsbelegung \mathbf{x}_1
Bestimme mit derandomisiertem LP-Runden eine Wahrheitsbelegung \mathbf{x}_2
Gib die bessere der beiden Wahrheitsbelegungen aus

Satz 9.5 *Der deterministische „Best of two“ Algorithmus für MAX SAT hat eine Approximationsgarantie von $3/4$.*

Beweis. Wir nehmen zunächst an, dass wir mit Wahrscheinlichkeit je $1/2$ entscheiden, ob wir \mathbf{x}_1 oder \mathbf{x}_2 ausgeben. Seien wieder W und W_c , $c \in \mathcal{C}$, Zufallsvariablen wie im Beweis von Satz 9.1. Für Klauseln c mit $|c| = k$ gilt wegen (9.1) und $z_c^* \leq 1$

$$\mathbf{E}[W_c \mid \text{Ausgabe von } \mathbf{x}_1] = \alpha_k w_c \geq \alpha_k w_c z_c^* .$$

Nach (9.4) gilt

$$\mathbf{E}[W_c \mid \text{Ausgabe von } \mathbf{x}_2] \geq \beta_k w_c z_c^*$$

und zusammen

$$\mathbf{E}[W_c] = \frac{1}{2}(\mathbf{E}[W_c \mid \text{Ausgabe von } \mathbf{x}_1] + \mathbf{E}[W_c \mid \text{Ausgabe von } \mathbf{x}_2]) \geq w_c z_c^* \frac{\alpha_k + \beta_k}{2} .$$

Nun ist $\alpha_1 + \beta_1 = \alpha_2 + \beta_2 = 3/2$ und für $k \geq 3$ gilt $\alpha_k + \beta_k \geq 7/8 + (1 - 1/e) \geq 3/2$ und daher $\mathbf{E}[W_c] \geq 3/4 \cdot w_c z_c^*$. Wieder können wir das erwartete Gewicht der ausgegebenen Wahrheitsbelegung abschätzen als

$$\mathbf{E}[W] = \sum_{c \in \mathcal{C}} \mathbf{E}[W_c] \geq \frac{3}{4} \sum_{c \in \mathcal{C}} w_c z_c^* = \frac{3}{4} UB \geq \frac{3}{4} OPT . \quad (9.5)$$

Eine der bedingten Erwartungen $\mathbf{E}[W_c \mid \text{Ausgabe von } \mathbf{x}_1]$ oder $\mathbf{E}[W_c \mid \text{Ausgabe von } \mathbf{x}_2]$ muss wenigstens $\mathbf{E}[W]$ groß sein. Daher ist das Gewicht aller erfüllten Klauseln der besseren der beiden Wahrheitsbelegungen \mathbf{x}_1 oder \mathbf{x}_2 wenigstens $\mathbf{E}[W] \geq \frac{3}{4} OPT$, was zu beweisen war. \square

Nach (9.5) ist $\mathbf{E}[W] \geq \frac{3}{4} UB$. Die ganzzahlige Lösung des „Best of two“ Algorithmus hat Zielfunktionswert wenigstens $\mathbf{E}[W]$. Wir erhalten daher

Korollar 9.6 *Die Ganzzahligkeitslücke des linearen Programms (9.3) ist höchstens $4/3$.*

Beispiel. Sei $f = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$ mit Einheitsgewichten für jede Klausel. Setzen wir $y_1 = \dots = y_4 = \frac{1}{2}$, können alle $z_c = 1$, $c \in \mathcal{C}$, gesetzt werden, woraus wir $UB = 4$ erhalten. Allerdings gilt $OPT = 3$, weil sich die erste und die letzte sowie die mittleren beiden Klauseln gegenseitig ausschließen. Für diese Instanz ist die Ganzzahligkeitslücke des linearen Programms (9.3) genau $4/3$.

Tight example

Unsere Analyse des „Best of two“ Algorithmus ist scharf, wie das folgende Beispiel zeigt. Sei $f = (x_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3)$ mit den Gewichten 1, 1 und $2 + \varepsilon$ auf den Klauseln (in dieser Reihenfolge). Weil alle Klauseln genau zwei Literale enthalten, werden die Variablen sowohl bei randomisiertem Runden als auch bei randomisiertem LP-Runden mit Wahrscheinlichkeit $1/2$ auf *wahr* gesetzt. Sei x_1 die Variable, die in der Derandomisierung zuerst fixiert wird. Die bedingten Erwartungen sind $\mathbf{E}[W \mid x_1 = \textit{wahr}] = 3 + \varepsilon/2$ und $\mathbf{E}[W \mid x_1 = \textit{falsch}] = 3 + \varepsilon$. Daher wird $x_1 = \textit{falsch}$ gesetzt. Dies führt auf ein Gesamtgewicht aller erfüllten Klauseln von $3 + \varepsilon$, wohingegen bei Belegung von x_1 mit *wahr* ein Gewicht von $4 + \varepsilon$ möglich ist.

Die zurzeit beste erreichte Güte für MAX SAT ist 0,8331 [6]. Wir werden im nächsten Kapitel sehen, dass das Problem Eigenschaften hat (die zur Definition der Klasse Max \mathcal{SNP} führen), wegen denen kein PTAS für MAX SAT existieren kann.

Kapitel 10

Hardness of Approximation

last edit:
13.2.07

Der Titel dieses Kapitels ist ein feststehender Begriff. Wir verstehen darunter die Tatsache, dass das Auffinden einer approximativen Lösung genauso schwer sein kann wie das Bestimmen einer optimalen Lösung selbst. In den letzten 15 Jahren hat hier eine überraschende Entwicklung neuer Charakterisierungen von Komplexitätsklassen eingesetzt, die das Klassifizieren von \mathcal{NPO} -Problemen bezüglich ihrer Approximierbarkeit ermöglicht, s. etwa [3].

Ein informeller Zugang zur Klasse \mathcal{NP}

Folgendes kann man etwa bei Garey und Johnson [12] nachlesen. Ein Entscheidungsproblem hat als Lösung die Antwort *ja* oder *nein*. Die Instanzen mit Antwort *ja* heißen *Ja-Instanzen*, die mit Antwort *nein* heißen *Nein-Instanzen*. Ein *nichtdeterministischer Algorithmus* \mathcal{A} für ein Entscheidungsproblem Π besteht aus zwei Stufen, *Raten* und *Überprüfen* (*guess and verify*). Zu gegebener Instanz $I \in \Pi$ rät \mathcal{A} einen möglichen *Beweis* S der Zugehörigkeit I 's zu den Ja-Instanzen. I und S werden der zweiten Stufe, dem *Verifier*, übergeben, der mit Antwort *ja* oder mit Antwort *nein* terminiert oder nicht terminiert. Man sagt, ein solcher Algorithmus \mathcal{A} *löst* Π , wenn für alle Instanzen $I \in \Pi$ gilt

1. falls I eine *Ja-Instanz* ist, gibt es einen Beweis S so, dass der Verifier von \mathcal{A} die Antwort *ja* gibt
2. falls I eine *Nein-Instanz* ist, gibt es keinen Beweis S so, dass der Verifier von \mathcal{A} die Antwort *ja* gibt

Wird im ersten Fall die Antwort stets in polynomialer Zeit in $|I|$ gegeben, so heißt \mathcal{A} *nichtdeterministisch polynomial*. Insbesondere impliziert dies, dass $|S|$ nur polynomiale Größe haben darf. Zur Klasse \mathcal{NP} gehören alle Entscheidungsprobleme, die mit einem nichtdeterministisch polynomialen Algorithmus gelöst werden können. Man sagt auch, \mathcal{NP} enthält genau diejenigen Entscheidungsprobleme mit einem deterministisch polynomial überprüfbareren *Ja-Zertifikat*. Eine weitere Sprechweise: Es gibt einen kurzen Beweis der Zugehörigkeit zu \mathcal{NP} .

Beispiel. Die Entscheidungsversion des TSP (gibt es zu einem gegebenen Graph mit rationalen Kantengewichten eine Tour der Länge k oder kürzer?) ist in \mathcal{NP} , denn wir *raten* eine

beliebige Reihenfolge S der n Städte (S ist offenbar polynomial in n) und können in Polynomialzeit überprüfen, ob die Länge der resultierenden Tour k nicht überschreitet. Anders als bei deterministisch polynomial lösbaren Problemen (denen in \mathcal{P}), gibt es kein polynomial überprüfbares *Nein-Zertifikat*: Die Frage nach der *Nicht-Existenz* einer Tour der Länge k oder kürzer kann *prinzipiell* nicht anders als durch Enumeration aller Touren beantwortet werden, es sei denn $\mathcal{P} = \mathcal{NP}$.

10.1 PCP Theorem

In obiger Definition der Klasse \mathcal{NP} wird der Beweis in seiner gesamten Länge geprüft. Als Beispiel diene das SAT Problem und als Beweis der Erfüllbarkeit einer Formel eine gegebene Wahrheitsbelegung. Zum Nachweis, dass diese Belegung tatsächlich erfüllend ist, muss die Wahrheitsbelegung jeder Variablen überprüft werden. Ein einzelnes Bit des Beweises kann möglicherweise den Ausschlag darüber geben, ob ein Beweis akzeptiert wird oder nicht. Diese Situation ist im Kontext approximativer Lösungen nicht befriedigend: Wann ist ein Beweis „annähernd richtig“? Man erlaubt daher die mächtige Erweiterung, dass der Verifier mit Hilfe eines Strings *zufälliger* Bits nur einen Teil der Bits des Beweises auswählt, die dann gelesen werden. Auf Basis des Gelesenen wird eine deterministische Entscheidung getroffen.

Definition 10.1 Ein $(r(n), q(n))$ -Verifier darf bei einer Instanz der Länge n für seine Auswahl höchstens $O(r(n))$ zufällige Bits benutzen und höchstens $O(q(n))$ Bits des Beweises lesen.

Definition 10.2 Für ein Entscheidungsproblem Π überprüft ein Verifier einen Beweis probabilistisch, falls

1. für jede Ja-Instanz $I \in \Pi$ gibt es einen Beweis so, dass der Verifier für jede zufällige Auswahl von Bits die Antwort ja gibt (d.h. mit Wahrscheinlichkeit 1),
2. für jede Nein-Instanz $I \in \Pi$ gibt der Verifier für jeden Beweis mit Wahrscheinlichkeit wenigstens $\frac{1}{2}$ die Antwort nein aus.

Die Wahrscheinlichkeit ist hier über alle zufälligen binären Strings der Länge $O(r(n))$.

Definition 10.3 Die Klasse $PCP(r(n), q(n))$ besteht aus den Problemen, deren Beweise mit einem $(r(n), q(n))$ -Verifier probabilistisch überprüft werden können.

PCP steht für *probabilistically checkable proofs*. Offenbar gilt $\mathcal{NP} = PCP(0, \text{poly}(n))$: Der Verifier benutzt keine zufälligen Bits und überprüft eine polynomiale Anzahl von Bits des Beweises (nämlich alle Bits, für Probleme in \mathcal{NP} hat der Beweis nur polynomiale Länge). Weitاً überraschender sind einige weitere probabilistische Charakterisierungen der Klasse \mathcal{NP} , am bekanntesten das sehr tief liegende PCP Theorem [4, 5] von 1992:

Satz 10.4 $\mathcal{NP} = PCP(\log n, 1)$.

Abbildung 10.1 skizziert dieses Ergebnis. Zu gegebener Instanz I und einem Beweis x polynomialer Länge (in $|I|$) wirft der Verifier $O(\log n)$ Münzen (mit $n = |I|$) um $k = O(1)$

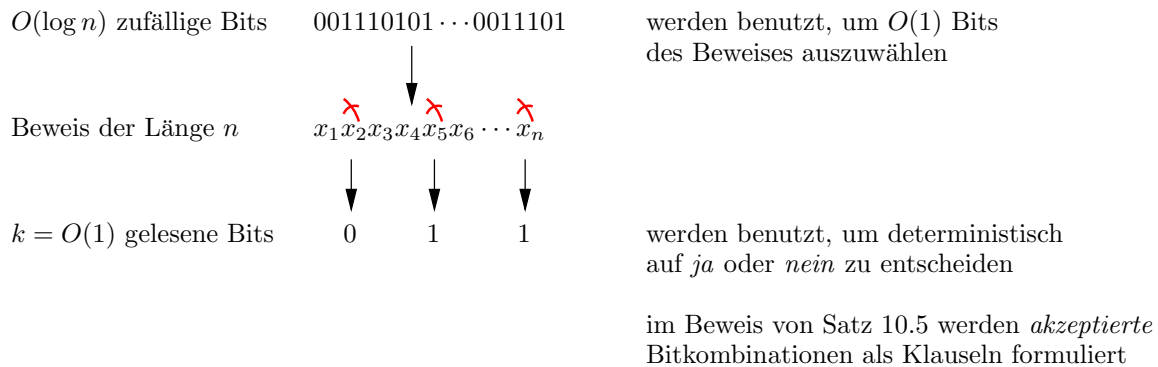


Abbildung 10.1: Zur Funktionsweise eines $(\log n, 1)$ -Verifiers

Bits in x zu bestimmen, die überprüft werden. Basierend auf den Werten dieser k Bits entscheidet der Verifier auf *ja* oder *nein*. Für eine Ja-Instanz gibt es einen Beweis so, dass die Antwort *ja* lautet, unabhängig von den zufälligen Bits und für eine Nein-Instanz wird mit Wahrscheinlichkeit wenigstens $1/2$ *nein* geantwortet. Satz 10.4 besagt, dass es einen solchen Verifier für Probleme in \mathcal{NP} tatsächlich gibt.

Lücken erzeugen mit dem PCP Theorem

Beweise der \mathcal{NP} -Vollständigkeit lassen selten eine Aussage über die Zielfunktionswerte der verwendeten Instanzen zu, daher sind (Nicht-)Approximierbarkeitsresultate kaum übertragbar. Zum Beispiel ist der Beweis von Satz 2.1 eine Ausnahme. Wir erzwingen zwei Bereiche möglicher Zielfunktionswerte, die durch eine *Lücke* voneinander getrennt werden. Das PCP Theorem gibt uns eine allgemeine Technik, eine solche Lücke zu erzeugen.

Satz 10.5 *Falls nicht $\mathcal{P} = \mathcal{NP}$, gibt es ein $\varepsilon > 0$ so, dass es keinen $(1 - \varepsilon)$ -Approximationsalgorithmus für MAX 3SAT gibt.*

Beweis. Sei ein \mathcal{NP} -vollständiges Problem Π und eine Instanz $I \in \Pi$ gegeben. Nach PCP Theorem gibt es einen Verifier, der $O(\log n)$ zufällige Bits benutzen und konstant viele Bits des Beweises überprüfen darf. Die Zahl der möglichen Kombinationen der $O(\log n)$ zufälligen Bits ist $S = 2^{O(\log n)}$, also polynomial in n . Seien daraufhin die k Bits, die aus dem Beweis gelesen werden i_1, \dots, i_k . Der Verifier gründet seine deterministische Entscheidung auf diese k Bits.

Wir können die Bedingungen, dass der Verifier den Beweis akzeptiert als Boolesche Formel in den n Variablen der Bits des Beweises aufschreiben. Jede Belegung der k gelesenen Bits, die auf *akzeptieren* führt, ergibt eine Konjunktion (und) von k Literalen, d.h. eine Klausel der Länge k . Diese werden disjunktiv (oder) miteinander verknüpft. Equivalent können wir diese Formel in konjunktive Normalform bringen, d.h. als Konjunktion von Disjunktionen schreiben. Diese k SAT-Formel hat höchstens 2^k Klauseln. Weil k konstant ist, kann sie als 3SAT-Formel mit einer konstanten Anzahl von Variablen und Klauseln geschrieben werden (die Anzahl hängt exponentiell von k ab; benutzt man die Standardreduktion von SAT auf 3SAT [12] so erhält man höchstens $k2^k$ Klauseln und Variablen). Diese konstante Anzahl Klauseln sei M .

Falls I eine Ja-Instanz ist, wissen wir, dass es einen Beweis gibt, der für jede Wahl zufälliger Bits akzeptiert wird. Das heißt, dass für alle S Belegungen der zufälligen Bits alle jeweils daraufhin gebildeten M Klauseln erfüllbar sind. Andererseits, falls I eine Nein-Instanz ist, muss jeder Beweis bei wenigstens der Hälfte der S möglichen Belegungen zufälliger Bits auf *nicht akzeptieren* führen. Das heißt, für wenigstens die Hälfte der Zufallsbelegungen können die jeweils zugehörigen Klauseln nicht alle gleichzeitig erfüllt sein. Es muss demnach wenigstens $S/2$ Klauseln geben, die nicht erfüllbar sind.

Das MAX 3SAT Problem auf allen SM Klauseln hat daher zwei wesentlich verschiedene Bereiche von Zielfunktionswerten: Für eine Ja-Instanz von Π ist $OPT = SM$, für eine Nein-Instanz ist $OPT \leq SM - S/2$. Im ersten Fall liefert ein $(1 - \varepsilon)$ -Approximationsalgorithmus für MAX 3SAT mit $\varepsilon < 1/2M$ als Zielfunktionswert wenigstens

$$(1 - \varepsilon) \cdot OPT > \frac{2M - 1}{2M} \cdot SM = SM - \frac{S}{2} .$$

Wir können in diesem Fall schließen, dass dann alle SM Klauseln erfüllbar sein müssen. Im zweiten Fall kann der approximative Zielfunktionswert selbstverständlich nicht besser als $OPT = SM - S/2$ sein. Somit kann ein solcher Approximationsalgorithmus in Polynomialzeit entscheiden, ob I eine Ja-Instanz für Π ist oder nicht, ein Widerspruch zu $\mathcal{P} \neq \mathcal{NP}$. \square

Etwa für das NP-vollständige Problem SAT bedeutet dieser Satz folgendes. Es gibt ein festes $\varepsilon > 0$ und eine polynomiale Reduktion τ von SAT auf MAX 3SAT so, dass für jede Boolesche Formel f

$$\begin{aligned} f \text{ ist erfüllbar} &\implies \text{MAX 3SAT}(\tau(f)) = 1 \\ f \text{ ist nicht erfüllbar} &\implies \text{MAX 3SAT}(\tau(f)) < \frac{1}{1 + \varepsilon} \end{aligned}$$

Die Zielfunktionswerte von MAX 3SAT werden hier als *Anteil* der erfüllten Klauseln dargestellt, d.h. normiert mit $\sum_{c \in \mathcal{C}} w_c$. Eine Reduktion wie in Satz 10.5 nennen wir eine *Lücken erzeugende Reduktion*. Am Ende des Beweises ist klar geworden, wie man diese Lücke einsetzt: Liegt der Zielfunktionswert einer approximativen Lösung in der Lücke, so ist die approximative Lösung bereits optimal, s. Abbildung 10.2.

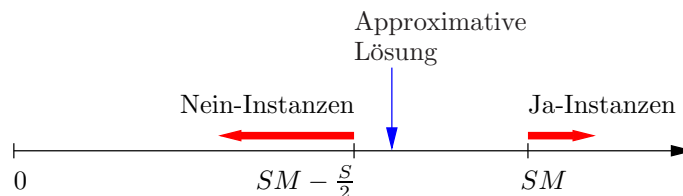


Abbildung 10.2: Weil zwischen den zu Nein-Instanzen gehörenden Zielfunktionswerten und den zu Ja-Instanzen gehörenden Zielfunktionswerten eine echte *Lücke* ist, kann für jede approximative Lösung, deren Zielfunktionswert genau in die Lücke fällt geschlossen werden, dass die betrachtete Instanz eine Ja-Instanz sein muss.

Lücken bewahrende Reduktionen

Könnten wir die soeben erzeugte Lücke in Reduktionen bewahren, ließe sich Nichtapproximierbarkeit auf anderen Probleme übertragen.

Definition 10.6 Seien Π und Π' Maximierungsprobleme. Eine Lücken bewahrende Reduktion von Π auf Π' mit Parametern (c, ρ) , (c', ρ') ist ein polynomialer Algorithmus τ , der für jede Instanz $I \in \Pi$ eine Instanz $\tau(I) = I' \in \Pi'$ erzeugt. Für die Optima gelten folgende Bedingungen

$$\begin{aligned} OPT(I) \geq c &\implies OPT(I') \geq c' \\ OPT(I) < \frac{c}{\rho} &\implies OPT(I') < \frac{c'}{\rho'} \end{aligned}$$

Hierbei sind c und ρ Funktionen von $|I|$ und c' und ρ' Funktionen von $|I'|$. Außerdem gilt $\rho(I), \rho'(I') \geq 1$.

Bemerkung. Diese Definition kann man so anpassen, dass auch eines (oder beide) der Probleme Minimierungsprobleme sind. Die Größe der Lücke wird nicht erhalten, nur ihre Existenz. Man beachte, dass eine solche Reduktion lediglich erlaubt, aus der Nicht-Approximierbarkeit eines Problems Π die Nicht-Approximierbarkeit eines Problems Π' zu schließen. Ein Approximationsalgorithmus für Problem Π' führt nicht auf einen Approximationsalgorithmus für Π .

last edit:
14.2.07

10.2 Die Klasse Max \mathcal{SNP}

Warum lassen sich, wie für MAX 3SAT, für viele Probleme keine polynomialen Approximationsschemata finden? Welche Probleme verhalten sich bezüglich ihrer Approximierbarkeit wie MAX 3SAT? Papadimitriou und Yannakakis [34] beantworteten diese Fragen mit der Definition einer neuen Komplexitätsklasse Max \mathcal{SNP} . Der Versuch, die Nicht-Approximierbarkeit von Problemen in dieser Klasse zu beweisen, hat später zur Entdeckung des PCP-Theorems geführt.

Max \mathcal{SNP} ist formal über die logische Charakterisierung von \mathcal{NP} definiert. Wir betrachten ein Beispiel, das uns eher intuitiv an diese Klasse heranführen soll. Das Problem SAT kann man folgendermaßen ausdrücken:

$$\exists T \forall c \exists x : [(P(c, x) \wedge x \in T) \vee (N(c, x) \wedge x \notin T)]$$

T sei hierbei die Menge der wahren Variablen, die binären Relationen P und N kodieren die Instanz: $P(c, x)$ ist wahr bedeutet, dass Variable x in Klausel c positiv auftaucht, entsprechend $N(c, x)$ für negierte Variablen x in c .

3SAT lässt sich ohne den zweiten Existenzquantor schreiben, wenn wir die Instanz mit vier ternären Relationen C_0, C_1, C_2, C_3 beschreiben, wobei $C_j(x_1, x_2, x_3)$ wahr ist, wenn genau j der Variablen negiert sind: Wir schreiben dann

$$\begin{aligned} \exists T \forall (x_1, x_2, x_3) : & [((x_1, x_2, x_3) \in C_0 \implies x_1 \in T \vee x_2 \in T \vee x_3 \in T) \wedge \dots \wedge \\ & ((x_1, x_2, x_3) \in C_3 \implies x_1 \notin T \vee x_2 \notin T \vee x_3 \notin T)] \end{aligned}$$

Sehr grob gesagt besteht die Klasse \mathcal{SNP} der *strikten* \mathcal{NP} Probleme aus den Problemen, die sich so wie 3SAT schreiben lassen als $\exists S \forall x : \phi(G, S, x)$ mit Relationen G und S und einer quantorenfreien Formel $\phi(G, S, x)$, wobei G und S zur Kodierung der Instanz und einer Lösung benutzt werden. Dies wird, wie im Falle des 3SAT Problems möglich, wenn sich die Bedingungen an die gesuchte Struktur instanzunabhängig enumerieren lassen (für SAT funktioniert das nicht, weil Klauseln beliebig lang werden können). Zu jedem \mathcal{SNP} Problem kann man ein Maximierungsproblem definieren als $\max_S |\{x \mid \phi(G, S, x) = \text{wahr}\}|$; etwas vereinfacht bilden diese Probleme die Klasse $\text{Max } \mathcal{SNP}$. Nach unserem obigen Beispiel gilt

Lemma 10.7 $\text{MAX 3SAT} \in \text{Max } \mathcal{SNP}$.

Die Klasse $\text{Max } \mathcal{SNP}$ ist reich an Problemen, die uns bereits bekannt sind.

Lemma 10.8 $\text{MAX CUT} \in \text{Max } \mathcal{SNP}$.

Beweis. Für jeden Knoten sei eine Variable u gegeben und es sei G eine binäre Relation, die wir als *Adjazenz* (im graphentheoretischen Sinn) interpretieren. Es sei S eine unäre Relation, die wir als eine Farbklasse der Knoten interpretieren. Nun definieren wir

$$\phi(G, S, (u, v)) = (u < v) \wedge G(u, v) \wedge (S(u) \neq S(v))$$

und das MAX CUT Problem ist $\max_S \{(u, v) \mid \phi(G, S, (u, v)) = \text{wahr}\}$. □

Wir kennen für die beiden vorgenannten Probleme einen Algorithmus mit konstantem Approximationsfaktor. Dies ist kein Zufall [34].

Satz 10.9 *Für jedes Problem in $\text{Max } \mathcal{SNP}$ gibt es einen α -Approximationsalgorithmus mit α konstant, d.h. $\text{Max } \mathcal{SNP} \subseteq \mathcal{APX}$.*

Papadimitriou und Yannakakis haben eine in beiden Richtungen Approximierbarkeit bewahrende sogenannte *L-Reduktion* für die Klasse $\text{Max } \mathcal{SNP}$ eingeführt (L steht für linear), mit deren Hilfe sich der Begriff der Vollständigkeit eines Problems in dieser Klasse definieren lässt. Kann ein Problem Π' auf Π L-reduziert werden, so führt ein konstant-Faktor Approximationsalgorithmus für Π auf eine konstant-Faktor-Approximation für Π' . MAX SAT ist ein solches $\text{Max } \mathcal{SNP}$ -vollständiges Problem [32]. Benutzt man Lücken bewahrende Reduktionen (sie sind schwächer als L-Reduktionen), so definieren wir weniger restriktiv:

Definition 10.10 *Ein Maximierungsproblem Π heißt $\text{Max } \mathcal{SNP}$ -schwer, wenn für jedes Problem $\Pi' \in \text{Max } \mathcal{SNP}$ und jede zwei Konstanten $c \leq 1, \rho > 1$ zwei Konstanten $c' < 1, \rho' > 1$ existieren so, dass es eine Lücken bewahrende Restriktion von Π' auf Π mit Parametern (c, ρ, c', ρ') gibt. Die Zielfunktionswerte sind wie oben als Anteile interpretiert.*

Um die Nicht-Approximierbarkeit *aller* $\text{Max } \mathcal{SNP}$ -schweren Probleme zu zeigen reicht es, ein Problem $\Pi' \in \text{Max } \mathcal{SNP}$ und ein $\varepsilon > 0$ anzugeben, für das Approximation mit Güte $(1 - \varepsilon)$ \mathcal{NP} -schwer ist: Wir kennen bereits ein solches Problem: MAX 3SAT.

Satz 10.11 *Für jedes $\text{Max } \mathcal{SNP}$ -schwere Problem gibt es eine Konstante $c < 1$ so, dass das Erzielen der Approximationsgüte c \mathcal{NP} -schwer ist.*

Wir formulieren diesen Satz noch einmal um und erhalten die vielleicht wichtigste praktische Konsequenz aus der Klasse Max \mathcal{SNP} :

Satz 10.12 *Ein Max \mathcal{SNP} -schweres Problem hat kein PTAS, es sei denn $\mathcal{P} = \mathcal{NP}$ [4].*

Satz 10.9 und Satz 10.12 zusammen formuliert man auch so: Genau die Max \mathcal{SNP} -vollständigen Probleme verhalten sich bezüglich ihrer Approximierbarkeit wie MAX 3SAT. Gemeint ist hier insbesondere auch die Nicht-Approximierbarkeit. Ein weiteres Beispiel ist

Satz 10.13 *MAX CLIQUE ist Max \mathcal{SNP} -schwer.*

Beweis. MAX CLIQUE ist das Problem, zu gegebenem Graph G einen bezüglich Knotenzahl größten vollständigen induzierten Teilgraphen von G zu bestimmen.

Wir reduzieren Lücken bewahrend von MAX 3SAT. Sei f eine 3SAT-Formel mit m Klauseln in den Variablen x_1, \dots, x_n . Gegebenenfalls durch Wiederholung einzelner Literale stellen wir sicher, dass jede Klausel genau drei Literale enthält. Wir konstruieren einen Hilfsgraphen $\tau(f)$ mit $3m$ Knoten wie folgt: Jede Klausel wird durch ein Knotentripel dargestellt. Innerhalb einer Klausel verlaufen keine Kanten. Zwei Knoten in verschiedenen Klauseln sind adjazent genau dann, wenn die zugehörigen Literale *nicht* Negationen voneinander sind.

Jede Clique kann höchstens einen Knoten jedes Tripels enthalten. Weiterhin stehen keine zwei Knoten einer Clique für Negationen voneinander. Das heißt, dass zu gegebener Clique in natürlicher Weise eine Teilbelegung der Variablen abgelesen werden kann, die mindestens so viele Klauseln erfüllt, wie die Kardinalität ω der Clique ist. Anders herum betrachte aus jeder erfüllten Klausel zu gegebener Wahrheitsbelegung ein wahres Literal. Die zugehörigen Knoten bilden eine Clique, so dass die maximale Anzahl erfüllbarer Klauseln gleich der Kardinalität einer größten Clique ist. \square

Tatsächlich kann man wesentlich stärkere Nichtapproximierbarkeitsaussagen für MAX CLIQUE treffen [20].

Satz 10.14 *Wenn nicht $\mathcal{P} = \mathcal{NP}$, gibt es für MAX CLIQUE keinen $n^{1-\varepsilon}$ -Approximationsalgorithmus für festes $\varepsilon > 0$.*

Arora und Lund [3] teilen Probleme in vier Klassen bezüglich ihrer Nicht-Approximierbarkeit ein. Sie geben allerdings zu, dass diese Klassen weder „natürlich“ oder vollständig sind, sondern den momentanen begrenzten Wissensstand widerspiegeln. In Tabelle 10.1 gibt der *Faktor* an, welche Approximationsgüte zu erzielen \mathcal{NP} -schwer ist.

Bemerkung. Man findet in der Literatur auch oft \mathcal{APX} -schwere oder \mathcal{APX} -vollständige Probleme. Im „Kompendium“ [7] wird ähnlich zu den approximations-erhaltenden L-Reduktionen eine PTAS-Reduktion eingeführt (die die Existenz eines PTAS bewahrt). Die Kernaussage, die man aus der \mathcal{APX} -Schwere eines Problems Π zieht ist daher, dass Π wenigstens so schwer (bzgl. Approximation) ist wie alle Probleme in \mathcal{APX} , und es daher insbesondere kein PTAS für Π geben kann. In der praktischen Konsequenz haben daher \mathcal{APX} -schwer und Max \mathcal{SNP} -schwer dieselbe Bedeutung. Tatsächlich wurde gezeigt, dass eine „natürliche Verallgemeinerung“ von Max \mathcal{SNP} identisch zu \mathcal{APX} ist [27].

Klasse	Beispielproblem	Faktor
I	MAX 3SAT	0,974
II	SET COVER	$(1 - \delta) \log n$ für $\delta > 0$
III	LABEL COVER	$2^{\log^{1-\delta} n}$ für alle $\delta > 0$
IV	CLIQUE	n^ε für ein $\varepsilon > 0$

Tabelle 10.1: Nicht-Approximierbarkeits-Klassen nach Arora und Lund [3]

Kapitel 11

Über den Tellerrand...

last edit:
14.2.07

Approximationsalgorithmen sind innerhalb der diskreten Mathematik und theoretischen Informatik ein sehr aktives Forschungsgebiet. Wir haben einige der wichtigen Techniken kennen gelernt, insbesondere die sehr allgemein einsetzbaren LP-basierten. Bezüglich der Approximierbarkeit liegt die Komplexitätslandschaft wesentlich differenzierter vor uns, als es die pure Aussage der \mathcal{NP} -Vollständigkeit eines Problems erlaubt.

Natürlich haben wir einiges überhaupt nicht behandelt, z.B. den großen klassischen Bereich Scheduling. Aber das Einarbeiten in die ausgelassenen oder das Vertiefen der behandelten Gebiete sollte jetzt nicht mehr *zu* schwierig sein.

Vielleicht ja im Rahmen einer Diplomarbeit...?

Literaturverzeichnis

- [1] N. Alon, D. Moshkovitz und S. Safra. Algorithmic construction of sets for k -restrictions. In *ACM Transactions on Algorithms*, Band 2, S. 153–177, 2006.
- [2] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.
- [3] S. Arora und C. Lund. Hardness of approximations. In Hochbaum [22], Kapitel 10.
- [4] S. Arora, C. Lund, R. Motwani, M. Sudan und M. Szegedy. Proof verification and hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
- [5] S. Arora und S. Safra. Probabilistic checking of proofs: a new characterization of NP. *J. ACM*, 45(1):70–122, 1998.
- [6] T. Asano und D.P. Williamson. Improved approximation algorithms for MAX SAT. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, S. 96–105, 2000.
- [7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela und M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, Berlin, 1999.
- [8] R. Bar-Yehuda und S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2:198–203, 1981.
- [9] M. Bern und D. Eppstein. Approximation algorithms for geometric problems. In Hochbaum [22], Kapitel 8, S. 296–345.
- [10] V. Chvátal. *Linear Programming*. W.H. Freeman and Company, New York, 1983.
- [11] I. Dinur und S. Safra. The importance of being biased. In *Proceedings of the thirty-fourth annual ACM Symposium on Theory of Computing*, S. 33–42. ACM Press, 2002.
- [12] M.R. Garey und D.S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.
- [13] N. Garg, V.V. Vazirani und M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18:3–20, 1997.
- [14] M.X. Goemans und D.P. Williamson. New $3/4$ -approximation algorithms for the maximum satisfiability problem. *SIAM J. Discrete Math.*, 7:656–666, 1994.

- [15] M.X. Goemans und D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42:1115–1145, 1995.
- [16] M.X. Goemans und D.P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. In Hochbaum [22], Kapitel 4.
- [17] G.H. Golub und C.F. van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, 1989.
- [18] M. Grötschel, L. Lovász und A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, Berlin, 1988.
- [19] S. Guha und S. Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, S. 649–657, 1998.
- [20] J. Håstad. Some optimal inapproximability results. In *Proceedings of the twenty-ninth Annual ACM Symposium on the Theory of Computing*, S. 1–10, El Paso, Texas, 1997. ACM Press.
- [21] L. Heinrich-Litan und M.E. Lübbecke. Rectangle covers revisited computationally. In S.E. Nikolettseas, Herausgeber, *Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA05), Lect. Notes Comput. Sci.*, Band 3503, S. 55–66, Berlin, 2005. Springer-Verlag.
- [22] D.S. Hochbaum, Herausgeber. *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Co., Boston, MA, 1996.
- [23] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.
- [24] K. Jain und V.V. Vazirani. Approximation algorithms for metric facility location and k -median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM*, 48(2):274–296, 2001.
- [25] D.S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.
- [26] G. Karakostas, L. Caires, G.F. Italiano, L. Monteiro, Palamidessi C. und Yung M. A better approximation ratio for the vertex cover problem. In *ICALP: International Colloquium on Automata, Languages and Programming, Lect. Notes Comput. Sci.*, Band 3580, S. 1043–1050. Springer Berlin, 2005.
- [27] S. Khanna, R. Motwani, M. Sudan und U. Vazirani. On syntactic versus computational views of approximability. *SIAM J. Comput.*, 28(1):164–191, 1999.
- [28] S. Khot, G. Kindler, E. Mossel und R. O’Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs? In *Proceedings. 45th Annual IEEE Symposium on Foundations of Computer Science*, S. 146–154, 2004.

- [29] G. Lin, C. Nagarajan, R. Rajaraman und D.P. Williamson. A general approach for incremental approximation and hierarchical clustering. In *Proceedings of Annual ACM-SIAM Symposium on Discrete Algorithms*, S. 1147–1156, 2006.
- [30] J.S.B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k -MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999.
- [31] M.Mahdian, Y.Ye und J.Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, Lect. Notes Comput. Sci.*, Band 2462, S. 229–242, 2002.
- [32] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [33] C.H. Papadimitriou und S. Vempala. On the approximability of the traveling salesman problem. In *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, S. 126–133, 2000.
- [34] C.H. Papadimitriou und M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43:425–440, 1991.
- [35] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, Berlin, 2003.
- [36] V.V. Vazirani. *Approximation Algorithms*. Springer, Berlin, 2001.