RWTH Aachen **SS 2010**
Lehrgebiet Theoretische Informatik Klausur
Kneis—Langer—Rossmanith 26.7.2010

Effiziente Algorithmen – Gruppe A

Aufgabe 1A (10 Punkte)

Betrachten Sie das folgende Online-Problem Bin-Packing:

Zur Verfügung stehen unendlich viele Container der Kapazität $C \in \mathbf{N}$. Ihr Algorithmus bekommt in jedem Schritt i ein Objekt p_i der Größe $c_i \in \mathbf{N}$ und muss dieses Objekt sofort in einem der Container ablegen. Dabei darf die Belegung eines Containers seine Kapazität nicht überschreiten, wobei die Belegung eines Containers die Summe der Größen der Objekte in diesem Container ist. Objekte können nicht aufgeteilt werden. Einmal abgelegte Objekte dürfen nicht mehr bewegt werden. Das nächste Objekt erhält der Algorithmus erst, nachdem er das vorherige Objekt in einem der Container abgelegt hat.

Die Kosten einer Lösung sind die Anzahl der benutzten (nicht leeren) Container, die es zu minimieren gilt. Entwerfen Sie einen 2-kompetitiven Algorithmus und beweisen Sie diese Güte.

Lösungsansatz

Analog zur Tee-Aufgabe aus der Übung, werden wir jeden Container mindestens zur Hälfte füllen.

Der Algorithmus fügt dazu das Objekt p_i in den ersten Container ein, der schon mindestens ein Objekt enthält und noch genügend freien Platz für p_i enthält. Falls kein solcher Container existiert, wird p_i in einem neuen Container abgelegt.

Wir zeigen nun, daß zu jedem Zeitpunkt höchstens ein Container weniger als $L/2$ gefüllt ist. Angenommen die nicht-leeren Container l und k sind höchstens halb gefüllt.

Sei oBdA $l < k$. Da k nicht leer ist, enthält er ein Objekt p, das höchstens $L/2$ groß ist. Da aber $l < k$ ist und l noch mindestens $L/2$ viel Platz hat, hätte der Algorithmus das Objekt in l abgelegt und nicht in k. Dies ist ein Widerspruch zur Annahme, es gibt also keine zwei nicht leeren, höchstens halb gefüllte Container.

Da eine optimale Lösung die Container höchstens komplett füllen kann, ergibt sich nun sofort, daß unser Algorithmus höchstens $2|Opt| + 1$ Container belegt. Somit ist er 2-kompetitiv.

Typische Fehler

Fast immer wurde ein korrekter Algorithmus gefunden. Dies ist nicht weiter verwunderlich, da jeder Algorithmus, der Jobs versucht noch in Container unter zu bringen, bevor ein neuer Container geöffnet wird, die Güte zwei erreicht.

Oft konnte diese Güte aber nicht schlüssig bewiesen werden. Ein konkreter Vergleich zwischen optimaler Lösung (bzw einer unteren Schranke für die optimale Lösung) und erreichter Lösung (wie oben gezeigt) fehlte häufig.

In vielen Fällen wurde zudem argumentiert, daß eine gewisse Reihenfolge von bestimmten Jobs der "schlimmste Fall" wäre. Es wurde aber nie bewiesen, daß dies tatsächlich schlimmer als alle anderen Fälle ist. Somit wurde mit diesem Argument nur eine untere Schranke für die Güte gezeigt, nicht aber die geforderte obere Schranke.

Aufgabe 2A (10 Punkte)

Betrachten Sie das folgende Optimierungsproblem 3-Hitting Set:

Eingabe:	Eine Grundmenge $U = \{1, \ldots, n\}$, eine Familie $\mathcal{S} = \{S_1, \ldots, S_m\}$ von Teilmengen $S_i \subseteq U$ mit $\lvert S_i \rvert \leq 3$ für alle $1 \leq i \leq m$.
Zulässige Lösungen:	Eine Teilmenge $U' \subseteq U$, so daß U' jedes S_i abdeckt, d.h. $S_i \cap U' \neq \emptyset$ für jedes $1 \leq i \leq m$.
Frage:	Finden Sie zulässige Lösung U' mit minimaler Größe $\lvert U' \rvert$.

Finden Sie einen Algorithmus, der dieses Problem mit konstanter Güte approximiert. Beweisen Sie seine Korrektheit, Laufzeit und Güte.

Lösungsansatz

3-Hitting Set ist eine Verallgemeinerung von Vertex Cover. Letzteres kann man so formulieren, daß man aus der Grundmenge der Knoten V eine Teilmenge $U \subseteq V$ sucht, so daß jede Menge $e = \{v_i, v_j\} \in E$ abgedeckt wird. 3-Hitting Set ist also ein Vertex Cover in Hypergraphen, bei denen jede Kante höchstens drei Knoten berührt.

Somit ist es nun nahe liegend, den Algorithmus aus der Vorlesung an dieses Problem anzupassen.

Der Algorithmus wählt ein beliebiges $S_i \in \mathcal{S}$. Dann werden alle Elemente aus S_i der gesuchten Menge U' hinzugefügt und alle Menge S_j mit $S_i \cap S_j \neq \emptyset$ aus S entfernt. Dies wird solange wiederholt, bis $\mathcal{S}$ leer ist.

Die Laufzeit ist offensichtlich polynomiell, da in jeder Wiederholung mindestens eine Menge aus $\mathcal{S}$ entfernt wird und jeder einzelne Schritt nur polynomiell viele Operationen benötigt.

Ebenso ist der Algorithmus korrekt, da nur dann eine Menge S_j aus $\mathcal{S}$ entfernt wird, falls eines der Elemente aus S_j in U' eingefügt wird. Da der Algorithmus erst terminiert, wenn $\mathcal{S}$ keine Mengen mehr enthält, ist U' somit ein Hitting Set.

Die Hauptaufgabe besteht nun darin zu zeigen, daß dieser Algorithmus eine 3-Approximation liefert.

Dazu betrachten wir eine Menge $S_i = \{x_i, y_i, z_i\}$, die von unserem Algorithmus in Schritt i ausgewählt wird. Offensichtlich enthält die optimale Lösung mindestens eines dieser Elemente, oBdA x_i. Unser Algorithmus entfernt nun alle Mengen, die eines der Elemente aus $\{x_i, y_i, z_i\}$ enthalten.

Da insbesondere x_i entfernt wurde, werden also für jedes Element der optimalen Lösung höchstens zwei falsche Elemente in die Approximationslösung eingefügt, wir erhalten also analog zur Vorlesung eine 3-Approximation.

Typische Fehler

Für die Analyse der Güte ist es entscheiden, daß x_i in der Instanz nicht mehr auftritt. Es ist also fehlerhaft, wenn ein Element aus $\{x_i, y_i, z_i\}$ in die Lösung aufgenommen wird, und dann im nächsten Schritt vielleicht ein Element aus $\{x_i, y_i', z_i'\}$, und dann aus $\{x_i, y_i'', z_i''\}$ und so weiter. Die Größe der Mengen S_i hilft also nur dazu, die Kandidaten für die Approximation zu finden. Leider wurde dies sehr oft falsch angewendet, meist in der Art "Die Mengen enthalten nur drei Elemente, deshalb ist die Güte auch drei."

Ein offensichtliches Gegenbeispiel gegen dieses Argument ist übrigens, daß man sonst einfach ganz U nehmen könnte. Dies ist aber offensichtlich keine konstante Approximation.

Der Fehler trat meist in Kombination mit dem Entwurf eines Greedyalgorithmus als Lösung auf, welcher immer ein Element wählt, welches eine maximale Anzahl von Mengen trifft. In der Vorlesung wurde dies aus gutem Grund vermieden, da solch ein Greedyalgorithmus nicht besser als $\Theta(\log(n))$ sein kann.
Im folgenden geben wir kurz einen Beweis für diese Tatsache an. (Nur aus Gründen der Vollständigkeit, für eine korrekte Lösung der Klausur ist dies natürlich nicht relevant.) Dazu betrachten wir folgendes Beispiel: Sei U eine Menge von Knoten mit $|U| = n$. Wir wählen die Knotenmenge $V = U \cap \bigcup_{i=1}^{n} R_i$ mit $|R_i| = \lfloor \frac{n}{i} \rfloor$.
Jeder Knoten aus R_i wird nun mit genau i Knoten aus U verbunden, wobei kein Knoten aus U mit zwei Knoten aus R_i verbunden wird. Da R_i höchstens $|U|/i$ viele Knoten enthält, ist dies leicht möglich.
Somit haben Knoten aus U am Anfang höchstens Grad $|U|$, der einzige Knoten in R_n hat allerdings auch Grad $|U| = n$. Somit wählt ein Greedyalgorithmus vielleicht diesen Knoten aus und entfernt ihn. Im folgenden Schritt haben Knoten aus U noch höchstens Grad $n-1$, der Knoten in R_{n-1} aber auch. Per Induktion kann man leicht zeigen, daß in Schritt i die Knoten in U gerade höchstens Grad $n-i$ haben und jeder Knoten in R_{n-i} genau Grad $n-i$. Der Greedyalgorithmus findet also die Lösung $\bigcup_{i=1}^{n} R_i$.
Die Größe dieser Lösung ist

$$\sum_{i=1}^{n} |R_i| = \sum_{i=1}^{n} \left\lfloor \frac{n}{i} \right\rfloor \geq \sum_{i=1}^{n} \frac{1}{2} \frac{n}{i} \geq \frac{n}{2} H_n = \Theta(n \log(n)).$$

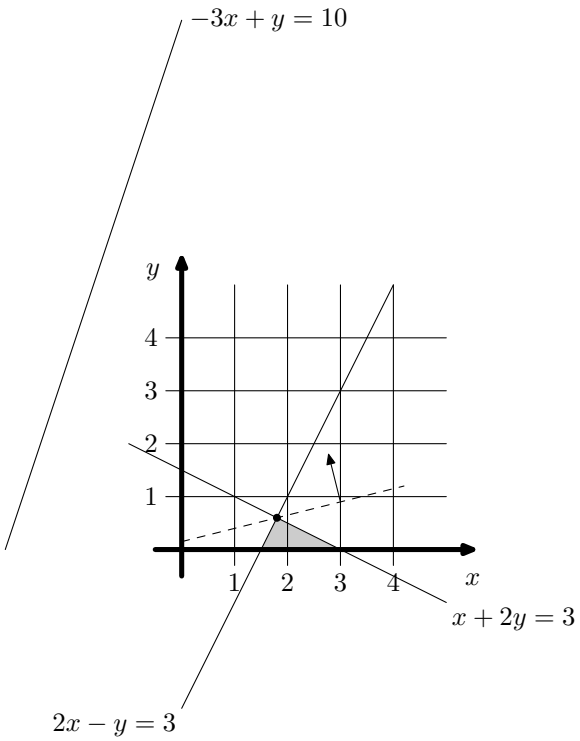
Optimal wäre natürlich U gewesen, mit Kosten n. Somit kann der Greedyalgorithmus natürlich auch für 3-Hitting Set keine konstante Güte garantieren.

Aufgabe 3A (10 Punkte)
Es sei das folgende Optimierungsproblem gegeben.
Maximiere $-x + 4y$ unter den Nebenbedingungen:

$$x + 2y \leq 3 \qquad -3x + y \leq 10$$
$$2x - y \geq 3 \qquad x, y \geq 0$$

Finden Sie eine optimale reelle und eine optimale ganzzahlige Lösung mittels eines Verfahrens Ihrer Wahl. Geben Sie Ihren Lösungsweg detailliert an.

Lösungsansatz
Für die reelle Lösung bietet sich eine grafische Lösung an. Bei Simplex muß man den Zweiphasenalgorithmus anwenden, da $(0, 0)$ keine zulässige Lösung ist. Zwar ist auch dieser recht schnell fertig, Graphisch läßt sich die Lösung jedoch sehr viel schnell finden.



Um den genauen Wert zu finden, müssen wir also entweder alle drei Ecken des Lösungsraums (hier: graues Dreieck) vergleichen oder aber die Zielfunktion so weit verschieben, daß sie auf dem Lösungsraum maximale Werte annimmt. Dies ist oben grafisch durch den Pfeil dargestellt.

Um den genauen Schnittpunkt zu erhalten, benötigen wir den Schnittpunkt der Geraden $x + 2y = 3$ und $2x - y = 3$, den man berechnen muß. Diesen ermittelt man leicht als $(1.8, 0.6)$. Vergleicht man den Wert der Zielfunktion mit dem der beiden anderen Ecken, sieht man leicht, daß dieser Schnittpunkt den optimalen Wert der Zielfunktion liefert, welcher 0.6 ist.

Da $(1.8, 0.6)$ offensichtlich nicht ganzzahlig ist, müssen wir nun noch die optimale ganzzahlige Lösung finden. Verwendet man Simplex, sollte man sich jetzt zwei neue LPs mit den zusätzlichen Einschränkungen $y \leq 0$ bzw. $y \geq 1$ erzeugen und diese dann getrennt lösen (vgl. das Verfahren aus der Vorlesung).

Man kann es aber viel schneller auch direkt lösen: Aus $x + 2y \leq 3$ folgt natürlich sofort $y \leq 1.5$, da $x \geq 0$ gilt. Aus $y = 1$ folgt aber einerseits $x + 2 \leq 3$ und $2x - 1 \geq 3$, also $x \leq 1$ und $2x \geq 4$. Es kann also keine Lösung mit $y = 1$ geben. Damit bleibt nur noch $y = 0$. Da aber in diesem Fall $x \leq 3$ und $2x \geq 3$ gilt, müssen wir nur die Punkte $(2, 0)$ und $(3, 0)$ überprüfen. Für ersteres erhalten wir den Wert -2 der Zielfunktion, für letzteren -3. Somit ist $(2, 0)$ die optimale ganzzahlige Lösung.

Typische Fehler

Gerne wurde Simplex angewendet, ohne auf den Zweiphasenalgorithmus zurückzugreifen. Mit Simplex konnte man die richtige Lösung dennoch meistens schnell finden, wenn man sich unterwegs nicht verrechnet hat.

Bei graphischen Lösungen wurden gerne die Halbebenen falsch herum eingezeichnet, es wurde also aus $\leq$ ein $\geq$ oder umgekehrt. Dies führte schnell dazu, daß der Lösungsraum leer schien, oder andere Punkt mit vermeintlich besserer Zielfunktion gefunden wurden (die aber natürlich gar nicht zulässig waren).

Oft wurden außerdem die Bedingungen $x, y \geq 0$ ignoriert, so daß der Lösungsraum als unbeschränkt eingezeichnet wurde.

Vielfach wurde nicht näher begründet, warum die Lösung $(1.8, 0.6)$ optimal ist.

Häufig wurde "ganzzahlige Lösung" falsch verstanden, und z.B. der Wert der Zielfunktion oder von x und y einfach ab- oder aufgerundet.

Aufgabe 4A (10 Punkte)

Entwerfen Sie einen effizienten Algorithmus für das folgende Problem. Gegeben sind ein ungerichteter Graph $G = (V, E)$, eine Kapazitätsfunktion $c\colon V \to \mathbf{N}$ und eine Knotenteilmenge $C \subseteq V$. Jeder Knoten $v \in C$ kann bis zu $c(v)$ der angrenzenden Kanten *abdecken*. Die Frage ist, ob auf diese Weise *alle* Kanten des Graphen abgedeckt werden können.

Geben Sie einen Beweis für die Korrektheit Ihres Algorithmus an, und zeigen Sie, daß seine Laufzeit polynomiell ist.

Lösungsansatz

Diese Aufgabe wurde schon als Hausaufgabe in Blatt 1 gestellt. Kopiert von dort:

Hier bietet sich wieder eine Modellierung als Flußproblem an: Wir konstruieren das Netzwerk (V', E', c') mit den Knoten $V' := \{s, t\} \cup E \cup C$ wie folgt:
$c'\big((s, e)\big) = 1$ für alle $e \in E$ und $c'\big((v, t)\big) = c(v)$ für $v \in C$. Außerdem setzen wir $c'\big((e, v)\big) = 1$ falls e in G inzident zu v ist.

Falls die Knoten aus C alle Kanten in G abdecken ohne ihre Kapazität zu überschreiten, kann jede Kante e einem Knoten $v(e)$ zugeordnet werden, der sie abdeckt. Es ist dann leicht, einen Fluß der Größe $|E|$ zu konstruieren, indem über jeden Pfad $s, e, v(e), t$ genau 1 fließt. Da alle Kanten in G abgedeckt werden, ohne die Kapazität der Knoten zu überschreiten, ist dies ein gültiger Fluß.

Laut Vorlesung ist in einem Netzwerk mit ganzzahligen Kapazitäten der maximale Fluß auch ganzzahlig. Falls es in diesem Netzwerk einen ganzzahligen Fluß der Größe $|E|$ gibt, so covert jeder Knoten v alle Kanten e für die $c'\big((e, v)\big) = 1$ gilt. Offensichtlich können so alle Kanten abgedeckt werden ohne die Kapazitäten der Knoten zu verletzen.

Typische Fehler

Überraschenderweise war der Lösungsweg vielen unbekannt. Es wurden daher oft kreative (aber leider falsche) Reduktionen auf das Flussproblem angegeben. Teilweise wurden auch immer noch Greedyalgorithmen eingesetzt.

Diejenigen, die einen korrekten Algorithmus angaben, kämpften mit den folgenden zwei Problemen:

Einerseits muss man zeigen, daß der Fluss genau dann $|E|$ ist, falls eine Lösung des Ursprungproblems existiert. Hier wurde oft nur eine Richtung gezeigt. Oft wurde auch versucht zu argumentieren, daß keine Lösung existieren kann, falls kein solcher Fluss existiert. Dies gelang aber fast nie, da dies viel schwieriger ist als aus der Existenz einer Lösung die Existenz eines Flusses (und anders herum) zu beweisen.

Zum anderen wurde oft übersehen, daß die Reduktion nur funktioniert, weil die Kapazitäten ganzzahlig sind und somit auch ein ganzzahliger Fluss existiert. Dieses Argument ist zwingend notwendig um aus dem Fluss das Cover zu konstruieren, da Flüsse sich andernfalls aufteilen können.