Institut für Informatik I
Prof. Dr. Heiko Röglin
Tobias Brunsch

universitätbonn

Algorithmen und Berechnungskomplexität III
Sommersemester 2011

Musterlösung Probeklausur

Aufgabe 1

(a) Ein r-Approximationsalgorithmus A für ein Minimierungsproblem Π ist ein Polynomialzeitalgorithmus, der für jede Instanz eine Lösung liefert, deren Kosten höchstens r-mal so groß sind wie die Kosten einer optimalen Lösung, d.h. $w_A(I) \leq r \cdot \mathrm{OPT}(I)$ für alle Instanzen $I \in \mathcal{I}_\Pi$. Die Kosten beziehen sich dabei auf die durch das Minimierungsproblem Π gegebene Zielfunktion.

(b) Die Eingabe für das metrische Traveling Salesperson Problem ist eine Menge V von Knoten und eine Metrik d auf V, d.h. $d(u,v) = 0 \iff u = v$, $d(u,v) = d(v,u)$ und $d(u,w) \leq d(u,v) + d(v,w)$ für alle $u, v, w \in V$. Gesucht ist eine Rundreise über alle Knoten von V mit minimalen Kosten.

(c) Ein Algorithmus ist pseudopolynomiell, wenn seine Laufzeit polynomiell in der Eingabelänge und in den in der Eingabe vorkommenden Zahlen beschränkt ist. Ein Problem ist stark $\mathcal{NP}$-hart, wenn es selbst für Instanzen, bei denen die in der Eingabe vorkommenden Zahlen polynomiell in der Eingabelänge beschränkt sind, $\mathcal{NP}$-hart ist.

(d) Eine Basislösung eines linearen Programms ist ein Vektor $x \in \mathbb{R}^d$, für den gilt:

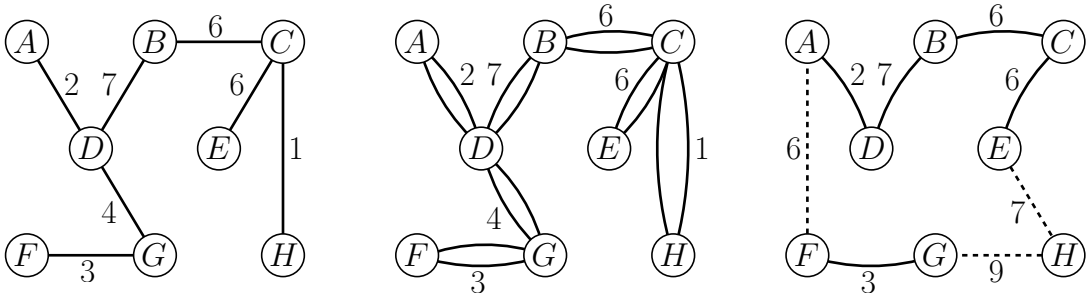
- Die zu den Nichtnulleinträgen von x gehörenden Spalten von A sind linear unabhängig und es gilt $Ax = b$. (Charakterisierung für Gleichungsform)
- Vektor x gehört zu mindestens d durch die Ungleichungsnebenbedingungen bestimmten Hyperebenen. (Charakterisierung für kanonische Form)

(e) Ein lineares Programm ist degeneriert,

- wenn es eine zulässige Basislösung mit weniger als m Nichtnulleinträgen gibt, wobei m die Anzahl der Gleichungsnebenbedingungen ist. (Charakterisierung für Gleichungsform)
- wenn es einen Punkt im Lösungspolyeder gibt, in dem sich mehr als d durch die Ungleichungsnebenbedingungen bestimmte Hyperebenen schneiden. (Charakterisierung für kanonische Form)

Aufgabe 2

(a) Wir bestimmen zunächst einen minimalen Spannbaum. In diesem Beispiel ist dieser eindeutig bestimmt (l) und besitzt Kosten 29. Dann verdoppeln wir die Kanten (m) und bestimmen einen Eulerkreis auf diesen Kanten. Ein möglicher Kreis ist $(A, D, B, C, E, C, H, C, B, D, G, F, G, D, A)$ mit Kosten $2 \cdot 29 = 58$. Durch Abkürzen (r) erhalten wir die Tour $(A, D, B, C, E, H, G, F, A)$ mit Kosten 46.



(b) Wir zeigen, dass das Problem HAMILTONIAN CYCLE in Polynomialzeit lösbar wäre, wenn es einen 2-Approximationsalgorithmus A für das allgemeine Traveling Salesperson Problem gäbe.

Sei $G = (V, E)$ ein ungerichteter Graph mit $n = |V|$ Knoten, den wir als Eingabe für das Problem HAMILTONIAN CYCLE erhalten. Als Distanzfunktion d betrachten wir folgende Funktion:

$$d(u,v) = \begin{cases} 0 & : \quad u = v\,, \\ 1 & : \quad \{u,v\} \in E\,, \\ 3n & : \quad \text{sonst}\,. \end{cases}$$

Anschließend führen wir Algorithmus A auf der Instanz (V, d) aus und akzeptieren, falls A eine Rundreise mit Kosten höchstens $2n$ liefert. Ansonsten verwerfen wir die Eingabe.

Zur Korrektheit: Besitzt G einen Hamiltonkreis, dann besitzt die optimale Rundreise Kosten von höchstens n, d.h. A wird eine Rundreise mit Kosten von höchstens $2n$ ausgeben und wir akzeptieren. Besitzt G keinen Hamiltonkreis, dann muss jede Rundreise eine „Nichtkante“ von G verwenden. Die optimale Rundreise und damit auch die von Algorithmus A bestimmte Rundreise haben dann Kosten von mindestens $3n$, d.h. wir verwerfen.

Aufgabe 3

(a) Wir verwenden dynamische Programmierung, um eine Tabelle T mit den Einträgen $T[i, \sigma_1, \sigma_2]$, $i = 0, \ldots, n$ und $\sigma_1, \sigma_2 = 0, \ldots, \sum_{i=1}^{n} b_i =: \sigma$ auszufüllen. Der Eintrag $T[i, \sigma_1, \sigma_2]$ gibt dabei an, ob es eine 3-Partition (I_1, I_2, I_3) der Indexmenge $\{1, \ldots, i\}$ gibt mit $\sum_{j \in I_1} b_j = \sigma_1$ und $\sum_{j \in I_2} b_j = \sigma_2$.

Als Konvention setzen wir $T[i, \sigma_1, \sigma_2] = 0$ für alle i, σ_1 und σ_2 mit $\sigma_1 < 0$ oder $\sigma_2 < 0$. Dann initialisieren wir die Tabelle für $i = 0$ mit

$$T[0, \sigma_1, \sigma_2] = \begin{cases} 1 & : \quad \sigma_1 = \sigma_2 = 0\,, \\ 0 & : \quad \text{sonst} \end{cases}$$

und berechnen die Einträge für $i = 1, \ldots, n$ sukzessive gemäß folgender Rekursionsformel:

$$T[i, \sigma_1, \sigma_2] = \begin{cases} 1 & : \quad T[i-1, \sigma_1 - b_i, \sigma_2] = 1\,, & \text{(Index } i \text{ gehört zu } I_1\text{)} \\ 1 & : \quad T[i-1, \sigma_1, \sigma_2 - b_i] = 1\,, & \text{(Index } i \text{ gehört zu } I_2\text{)} \\ 1 & : \quad T[i-1, \sigma_1, \sigma_2] = 1\,, & \text{(Index } i \text{ gehört zu } I_3\text{)} \\ 0 & : \quad \text{sonst}\,. & \text{(Es gibt keine entsprechende Partition.)} \end{cases}$$

Das ist wegen $\sigma \leq n \cdot B$ in Zeit $O(n \cdot \sigma^2) = O(n^3 \cdot B^2)$ möglich. Nun testen wir, ob σ durch 3 teilbar ist. Falls nicht, dann verwerfen wir die Eingabe. Ansonsten akzeptieren wir genau dann, wenn $T[n, \frac{\sigma}{3}, \frac{\sigma}{3}] = 1$ gilt.

(b) Wir verwenden eine Tabelle T mit Einträgen $T[i, w]$, wobei in $T[i, w]$ gerade der maximale Nutzen gespeichert ist, den man unter Verwendung der Objekte $1, \ldots, i$ und unter Einhaltung der Gewichtsschranke w erreichen kann. Die Pfeile geben an, wie die Einträge zu Stande kommen.

w \ i	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	17	17	17	17
2	0	0	0	17	17	17	23
3	0	36	36	36	36	36	40
4	0	36	41	53	53	53	53
5	0	36	41	58	58	58	59
6	0	36	41	58	64	65	76

Gemäß Eintrag $T[n, b]$ ist der maximal erreichbare Nutzen für die gegebene Instanz 76. Anhand der Pfeile können wir die zugehörige Lösung bestimmen: Die optimale Lösung verwendet die Objekte 1, 3 und 6.

Aufgabe 4

(a) Im Folgenden bezeichnen wir einen Eimer als *voll*, wenn er Objekte mit Gesamtgewicht mindestens 1 enthält. Nun betrachten wir den Algorithmus, der jedes Objekt in den ersten Eimer packt, der nicht voll ist. Dieser Algorithmus ist strikt 2-kompetitiv:

Zunächst stellen wir fest, dass zu keinem Zeitpunkt irgendein Eimer Objekte mit Gesamtgewicht von mindestens 2 enthält: Gäbe es einen solchen Eimer, so hätte dieser vor dem Hinzufügen des letzten Objektes bereits voll sein müssen, da jedes Objekt Gewicht kleiner 1 besitzt. Dann hätte obiger Algorithmus diesem Eimer aber kein Objekt mehr zugewiesen.

Es seien b Eimer durch obigen Algorithmus verwendet worden. Mit L_i bezeichnen wir das Gesamtgewicht der in Eimer i enthaltenen Objekte. Gemäß der Vorgehensweise obiges Algorithmus ist die Anzahl b^+ der vollen Eimer mindestens $b-1$. Die optimale Anzahl b^* voller Eimer ist höchstens $\sum_{j=1}^{n} w_j$. Somit erhalten wir folgende Ungleichungskette:

$$b^* \leq \sum_{j=1}^{n} w_j = \sum_{i=1}^{b} L_i = \sum_{i=1}^{b^+} L_i + \sum_{i=b^+ +1}^{b} L_i < 2b^+ + 1\,.$$

Aufgrund der Ganzzahligkeit folgt damit $b^* \leq 2b^+$, d.h. obiger Algorithmus ist strikt 2-kompetitiv.

(b) Wir führen die Annahme, dass es für ein $r < 2$ einen strikt r-kompetitiven Algorithmus A gibt, zu einem Widerspruch. Zunächst betrachten wir 2 Objekte, je mit Gewicht 3/4. Packt Algorithmus A beide Objekte in verschiedene Eimer, dann ist keiner der beiden Eimer voll. Optimal wäre in diesem Fall gewesen, beide Objekte in einen Eimer zu packen. Algorithmus A ist in diesem Fall nicht strikt kompetitiv.

Packt Algorithmus A beide Objekte in einen Eimer, dann präsentieren wir ihm anschließend zwei weitere Objekte, je mit Gewicht 1/4. Egal, wie sich der Algorithmus nun verhalten wird - genau ein Eimer wird am Ende voll sein. In der optimalen Lösung sind aber zwei Eimer voll. Daher kann Algorithmus A nicht strikt r-kompetitiv für $r < 2$ sein.

Aufgabe 5

(a) Das beschriebene Optimierungsproblem lässt sich wie folgt als lineares Programm modellieren:

$$\begin{aligned}
&\max 15x_1 + 20x_2 && \\
&x_1 + 2x_2 \leq 500 && \text{(Kapazitätsbeschränkung 1)} \\
&3x_1 + 4x_2 \leq 1500 && \text{(Kapazitätsbeschränkung 2)} \\
&x_1 + x_2 \leq 400 && \text{(Kapazitätsbeschränkung 3)} \\
&x_1 \leq 280, x_2 \leq 180 && \text{(Fertigungsbeschränkung)} \\
&x_1, x_2 \geq 0 &&
\end{aligned}$$

Die Koeffizienten der Zielfunktion ergeben sich gemäß der Formel

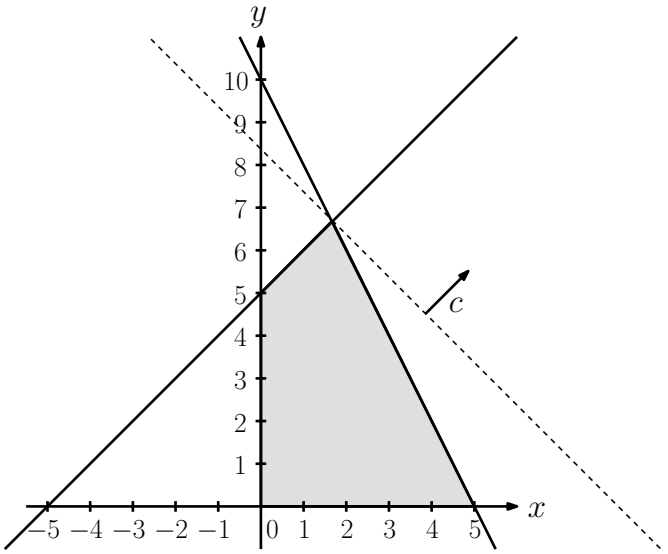
$$\text{Gewinn} = \text{Umsatz - Materialkosten - Fertigungskosten}\,.$$

(b) *Hinweis*: In der Klausur wird für graphisch zu lösende Aufgaben ein geeignet beschriftetes Koordinatensystem bereitgestellt werden.

Zunächst stellen wir die Gleichung $2x - 3y + z = 10$ nach z um und erhalten $z = -2x + 3y + 10$. Nun substituieren wir z gemäß dieser Formel sowohl in der Zielfunktion als auch in den Nebenbedingungen und erhalten das äquivalente Programm

$$\begin{aligned}
&\min -x - y + 10 \\
&2x + y \leq 10 \\
&-x + y \leq 5 \\
&x, y \geq 0
\end{aligned}$$

Statt die Funktion $-x - y + 10$ zu minimieren, können wir auch die Funktion $x + y$ maximieren. Beim Berechnen des optimalen Zielfunktionswertes müssen wir aber die ursprüngliche Zielfunktion wieder berücksichtigen.



Auf graphischem Weg erhalten wir als Lösung den Schnittpunkt der Hyperebenen, die durch die Gleichungen $2x + y = 10$ und $-x + y = 5$ definiert sind, also

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 10 \\ 5 \end{bmatrix} = \frac{1}{3} \cdot \begin{bmatrix} 1 & -1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 5 \end{bmatrix} = \begin{bmatrix} 5/3 \\ 20/3 \end{bmatrix}.$$

Als optimaler Zielfunktionswert ergibt sich damit $-5/3 - 20/3 + 10 = 5/3$.

(c) Wir verwenden im Folgenden die Schreibweise $x \in \{0, 1\}$ als Abkürzung für die Nebenbedingungen $x \geq 0$, $x \leq 1$ und $x \in \mathbb{Z}$. Nun können wir das Optimierungsproblem `CLIQUE` wie folgt als ILP formulieren:

$$\max \sum_{v \in V} x_v$$
$$\forall u \neq v \in V \text{ mit } \{u, v\} \notin E \colon x_u + x_v \leq 1$$
$$\forall v \in V \colon x_v \in \{0, 1\}$$

Die Variablen x_v sind Indikatorvariablen. Jede Lösung x entspricht einer Clique und umgekehrt. Die Nebenbedingungen stellen sicher, dass für jede Nichtkante $\{u, v\}$ eines Graphen nicht beide Knoten u und v für die Clique gewählt werden.