4 Approximationsalgorithmen und NP-harte Approximation

Ein *Approximationsalgorithmus* zu einem Optimierungsproblem Π ist ein polynomiell zeitbeschränkter Algorithmus A, der zu einer Eingabeinstanz I eine "legale Lösung" σ ausgibt, die eine "mathematische Gütegarantie" besitzt. Dabei sagen wir A hat *Güte* $k \geq 1$, wenn A zu jeder Eingabeinstanz I eine Lösung σ konstruiert, deren Wert vom Wert der optimalen Lösung multiplikativ maximal um Faktor k abweicht. Ein Approximationsalgorithmus der Güte 2 für ein Maximierungsproblem würde also stets mindestens die Hälfte des optimalen Profites erbeuten. Ein Approximationsalgorithmus der Güte 2 für ein Minimierungsproblem würde höchstens doppelt mal soviel Kosten wie nötig verursachen. Es ist naheliegend zu versuchen, ein *NP*-hartes Optimierungsproblem fast-optimal mit einem Approximationsalgorithmus zu lösen, dessen Güte möglichst nahe bei 1 liegen sollte. I.A. wird es eine Barriere k_0 geben, so dass Approximationsalgorithmen A_k mit Güte $k > k_0$ existieren, aber nicht mit Güte $k < k_0$ (unter der $P \neq NP$ Voraussetzung). Grenzfälle sind (formuliert für Minimierungsprobleme):

Keine konstante Gütegarantie $k_0 = \infty$.
Jeder Approximationsalgorithmus produziert Lösungen, deren Kosten die minimalen Kosten um einen beliebig großen Faktor überschreiten können.

Approximationsschema $k_0 = 1$.
Für jedes $k \geq 2$ existiert eine Approximationsalgorithmus A_ϵ mit Güte $1 + 1/k$.

Wir behandeln das Thema der Approximationsalgorithmen in diesem Abschnitt nur am Beispiel des Problems des Handelsreisenden und am Beispiel des Rucksackproblems. Es wird sich folgendes Bild ergeben:

- TSP besitzt keine konstante Gütegarantie (sofern $P \neq NP$).
- Für „Metrisches TSP" (ein Teilproblem von TSP) hingegen gibt es einen Approximationsalgorithmus der Güte 1.5.
- Für KNAPSACK gibt es ein Approximationsschema.

Am Ende des Abschnittes diskutieren wir kurz grundsätzliche Barrieren beim Design von Approximationsalgorithmen in Form von Resultaten zur NP-harten Approximation.

4.1 Approximierbarkeit von TSP

Für TSP (in seiner allgemeinen Form) kann man sich klar machen, dass keine konstante Gütegarantie existiert (außer wenn $P = NP$). Wir reduzieren zu diesem Zweck das Problem des Hamiltonschen Kreises (HC) in geeigneter Weise auf das Problem des Handelsreisenden (TSP). Sei $G = (V, E)$ der Eingabegraph zu HC. Knotenmenge V bestehe aus $n \geq 2$ Knoten, die wir mit den Nummern von 1 bis n identifizieren. Will man lediglich HC $\leq_{pol}$ TSP nachweisen, genügt die uns bereits bekannte Reduktion:

Setze in der $(n \times n)$-Distanzmatrix D den Eintrag $d_{i,j}$ auf 1 falls $\{i, j\} \in E$, und andernfalls auf 2.
Hieraus ergibt sich zwar die *NP*-Härte von TSP, aber es ist noch nicht ausgeschlossen, dass vernünftige Approximationsalgorithmen existieren. Betrachten wir aber nun die folgende leichte Modifikation der alten Reduktion:
Setze in der $(n \times n)$-Distanzmatrix D den Eintrag $d_{i,j}$ auf 1 falls $\{i, j\} \in E$, und andernfalls auf kn.
Es folgt, dass bez. D genau dann eine Rundreise der Länge n existiert, wenn G einen Hamiltonschen Kreis enthält. Falls jedoch in G kein Hamiltonscher Kreis existiert, muss die Rundreise mindestens eine Kante außerhalb von E verwenden. In diesem Fall hat sie mindestens die Länge $n - 1 + kn$. Gäbe es einen Approximationsalgorithmus für TSP der Güte k, so würde im Falle der Existenz eines Hamiltonschen Kreises in G eine Rundreise der Länge höchstens kn produziert, andernfalls jedoch eine Rundreise der Länge mindestens $kn + n - 1 > kn$. Mit anderen Worten: mit Hilfe der approximativen Lösung des TSP könnten wir HC exakt lösen. Falls $P \neq NP$, ist dies jedoch nicht möglich. Somit gibt es keine garantierte Güte k für Approximationsalgorithmen zu TSP (außer wenn $P = NP$).

Die Theorie der NP-Vollständigkeit hat uns signalisiert, dass es vermutlich Zeitverschwendung ist, nach einem Approximationsalgorithmus für TSP zu suchen. Betrachten wir nun aber die Einschränkung von TSP auf Distanzmatrizen D, die symmetrisch sind, d.h.,

$$\forall 1 \leq i < j \leq n : \; d_{ij} = d_{ji},$$

und die Dreiecksungleichung erfüllen, d.h.,

$$\forall 1 \leq i, j, k \leq n : \; d_{ik} \leq d_{ij} + d_{jk}.$$

Diese Einschränkungen sind für praktische Anwendungen durchaus vernünftig, denn sie besagen in salopper Formulierung:

Symmetrie Von A nach B ist es soweit wie von B nach A.

Dreiecksungleichung Von A nach C kann es nicht weiter sein als von A über B nach C.

Das durch Symmetriebedingung und Dreiecksungleichung eingeschränkte TSP-Problem wird als *Metrisches TSP* bezeichnet.

Eine Eingabe des metrischen TSP lässt sich auf die offensichtliche Weise als vollständiger, ungerichteter Graph mit Kantengewichten auffassen. Die Knoten repräsentieren die Städte und ein Kantengewicht die Distanz zwischen zwei Städten. Wegen obiger Symmetriebedingung können wir den Graphen als ungerichtet auffassen.

Wir stellen zur Bequemlichkeit ein paar (teilweise schon bekannte) graphentheoretische Konzepte bereit, die beim Entwurf eines Approximationsalgorithmus für das metrische TSP eine Rolle spielen. Sei G ein ungerichteter Graph. Ein *Weg* in G ist eine Folge $v_1, \ldots, v_r$ von $r \geq 1$ Knoten, wobei für alle $i = 1, \ldots, r-1$ Knoten v_i und v_{i+1} durch eine Kante verbunden sein müssen. (Ein Grenzfall ist der aus nur einem Knoten bestehende "Punktweg".) Falls $r \geq 2$ und $v_1 = v_r$, dann heisst der Weg auch *geschlossener Weg* oder *Kreis*. Eine *Euler-Tour* in G

ist ein Kreis, der jede Kante in G genau einmal durchläuft. G heisst *zusammenhängend*, wenn zwei Knoten sich stets durch einen Pfad miteinander verbinden lassen. Ein *ungerichteter Baum* ist ein zusammenhängender, kreisloser ungerichteter Graph. Wenn man aus einem Baum eine Kante entfernt (ohne die Randknoten der Kante dabei mitzuentfernen), zerfällt er in zwei Teile. Ein Baum ist gewissermaßen die ökonomischste Art, alle Knoten durch Pfade miteinander zu verbinden. Ein *Untergraph* von $G = (V, E)$ ist gegeben durch eine Knotenmenge $V' \subseteq V$ und alle Kanten aus E, die Knoten aus V' miteinander verbinden. Man spricht auch von dem *durch V' in G induzierten Untergraphen*. Ein *Teilgraph* von $G = (V, E)$ ist ein Graph $G' = (V', E')$ mit $V' \subseteq V$ und $E' \subseteq E$. Ein *Spannbaum (spanning tree)* von G ist ein Teilgraph der Form $T = (V, E')$, der ein Baum ist. Er enthält also alle Knoten von G und verbindet diese baumartig. Ein solcher Spannbaum kann natürlich nur dann existieren, wenn G zusammenhängend ist. Im Falle von Kantengewichten können wir $T = (V, E')$ die Kosten

$$c(T) = \sum_{e \in E'} w(e)$$

zuordnen. Ein *minimaler Spannbaum (minimum spanning tree)* ist ein Spannbaum minimaler Kosten. Es ist bekannt, dass minimale Spannbäume effizient berechnet werden können (zum Beispiel durch den Algorithmus von Kruskal).

Mit diesem Wissen ausgestattet ist es nun leicht, einen Approximationsalgorithmus der Güte 2 für das metrische TSP zu skizzieren. Es sei $G = (V, E)$ der vollständige ungerichtete Graph mit n Knoten $1, \ldots, n$. Knoten i repräsentiert dabei die i-te Stadt. Kante $\{i, j\}$ erhält als Gewicht die Distanz $d_{i,j}$ zwischen den Städten i und j. Wir berechnen einen minimalen Spannbaum $T = (V, E')$ von G (zum Beispiel mit dem Algorithmus von Kruskal). Seien c die Gesamtkosten von T. Wir erhalten eine Rundreise

$$R(T) = 3, 5, 8, 5, 1, 5, 7, 10, 7, 5, 3, 2, 9, 4, 9, 6, 9, 2, 3,$$

wenn wir (wie in Abbildung 1 angedeutet) einmal um T herumlaufen und dabei die angetroffenen Knoten (=Städte) der Reihe nach auflisten. Da Rundreise $R(T)$ jede Kante von T zweimal durchläuft, hat sie Länge $2c$. $R(T)$ hat noch einen Schönheitsfehler: die Städte werden i.A. mehrfach besucht. Wir erhalten eine legale Lösung von TSP — also eine Permutation $P(T)$ von $1, \ldots, n$ —, wenn wir in $R(T)$ alle Vorkommen von Knoten außer dem ersten (also alle Duplikate) streichen. In unserem Beispiel führt dies zu

$$P(T) = 3, 5, 8, 1, 7, 10, 2, 9, 4, 6.$$

Wegen der Dreiecksungleichung kann $P(T)$ nicht länger als $R(T)$ sein. Die Kosten von $P(T)$ sind also maximal $2c$.

Wie verhält es sich nun mit einer optimalen Rundreise? Im Graphen G formt diese einen Hamiltonschen Kreis C. Entfernen wir aus C (irgend-) eine Kante, entsteht ein (sehr spezieller) Spannbaum $T(C)$ (s. Abbildung 2). Die Länge von Rundreise C ist nicht kleiner als die Gesamtkosten von $T(C)$. Diese wiederum betragen mindestens c. Also hat C mindestens die Länge c und der skizzierte Approximationsalgorithmus die Güte 2.

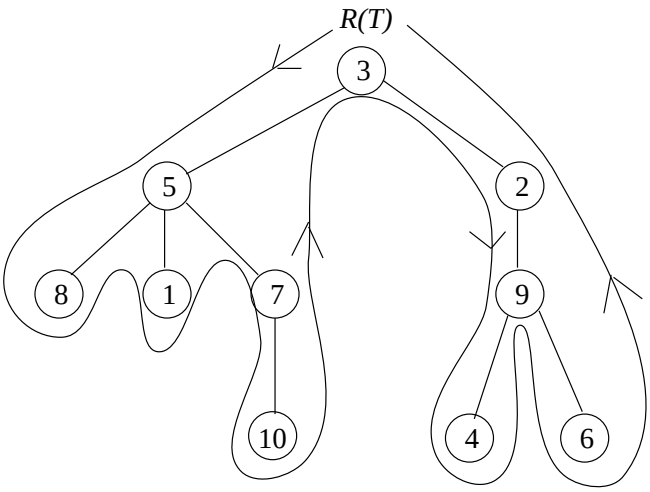

Abbildung 1: Die aus einem Minimum Spanning Tree abgeleitete Rundreise.

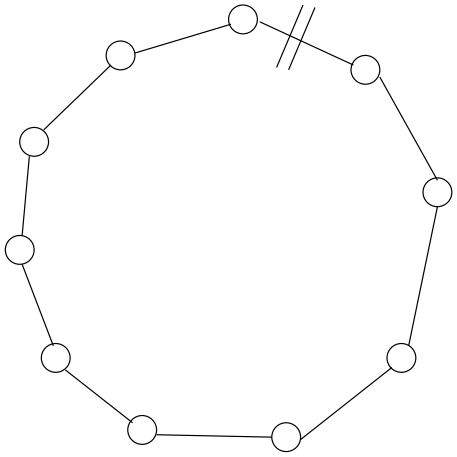

Abbildung 2: Der aus einer Rundreise abgeleitete Spanning Tree.

Vom bisherigen Erfolg berauscht stecken wir uns jetzt ein ehrgeizigeres Ziel: Entwurf eines Approximationsalgorithmus mit Güte 1.5. Zu diesem Zweck muss unser Approximationsalgorithmus noch etwas "aufgepeppt" werden. Der Faktor 2, der uns vom Optimum trennt, resultiert daraus, dass die konstruierte Rundtour jede Kante des minimalen Spannbaumes T implizit 2-mal durchläuft. Wenn wir eine Euler-Tour durch T zur Verfügung hätten, würden wir jede Kante nur 1-mal durchlaufen.

Dies wirft die Frage auf, für welche Graphen Euler-Touren existieren. Die Antwort liefert folgendes

Lemma 4.1 *Ein zusammenhängender Graph besitzt genau dann eine Euler-Tour, wenn jeder Knoten einen geraden Grad (also geradzahlig viele Nachbarn) besitzt.*[1]

[1]In diesem Fall spricht man auch von einem *Euler'schen Graphen.*

Beweis Wenn ein Knoten v mit ungeradem Grad, sagen wir Grad $2d+1$ existiert, dann kann es keine Euler-Tour geben: wenn der Kreis den Knoten v zum $d+1$-mal betritt, kann er ihn nicht mehr verlassen (Sackgassenargument).
Wenn ein zusammenhängender Graph G nur Knoten mit geradem Grad besitzt, dann kann eine Euler-Tour (effizient) konstruiert werden wie folgt:

1. Konstruiere einen ersten Kreis C, indem Du solange auf G spazieren gehst (ohne eine Kante doppelt zu laufen), bis Du zum Ausgangspunkt zurückkehrst.[2]

2. Wenn C bereits eine Euler-Tour darstellt, dann gib C aus. Andernfalls mache weiter mit Schritt 3.

3. Wähle einen Knoten v auf C, der noch unbenutzte Ausgangskanten hat.[3] Konstruiere von v aus einen zweiten Kreis C' (wieder durch „spazieren gehen“). Verschmelze[4] C und C' zu einem neuen Kreis und nenne diesen wieder C. Gehe zurück zu Schritt 2.

qed.

Aus dem Beweis ergibt sich die

Folgerung 4.2 *Es kann in Polynomialzeit getestet werden, ob ein Graph G eine Euler-Tour enthält. Gegebenenfalls kann diese auch in Polynomialzeit*[5] *konstruiert werden.*

Zurück zur Frage, ob es eine Euler-Tour durch den minimalen Spannbaum T gibt? Leider nein! In seiner Eigenschaft als Baum muss T Knoten ungeraden Gerades besitzen (zum Beispiel alle Blätter). Es gilt aber immerhin das folgende

Lemma 4.3 *Jeder Graph G hat geradzahlig viele Knoten ungeraden Grades.*

Beweis Wenn wir die Knotengrade $d_1, \ldots, d_n$ addieren zählen wir jede der m Kanten zweimal (da jede Kante zwei Randknoten besitzt):

$$\sum_{i=1}^{n} d_i = 2m \ .$$

Da die rechte Seite der Gleichung eine gerade Zahl ist, muss es auf der linken Seite geradzahlig viele ungeradzahlige Terme geben. **qed.**

[2] Da jeder Knoten geraden Grad hat, kann man den Spaziergang stets fortsetzen, solange man noch nicht den Ausgangspunkt erreicht hat.

[3] Da G zusammenhängend ist, muss es einen solchen Knoten v auf C geben.

[4] Laufe durch C bis zum Erreichen von v, dann durchlaufe C' bis zum erneuten Erreichen von v und schließlich durchlaufe den Rest des temporär unterbrochenen Kreises C.

[5] Auf einer „Random Access Maschine“ können wir „Polynomialzeit“ durch „Linearzeit“ präzisieren.

Idee Es seien $u_1, \ldots, u_{2k}$ die Knoten ungeraden Grades in T. Ergänze T zu einem Euler'schen Graphen, indem Du k „Heiratskanten" hinzufügst, die die Knoten u_i „perfekt verheiraten". Wähle hierzu k Heiratskanten mit minimalem Gesamtgewicht (ein sogenanntes „Minimum Weight Perfect Matching", welches in Polynomialzeit berechenbar ist).

Hieraus ergibt sich der folgende von Christofides vorgeschlagene Approximationsalgorithmus:

1. Gegeben die n-Clique (bestehend aus den n Städten) mit Kantengewichten $d_{i,j}$, berechne einen minimalen Spannbaum T.

2. Ergänze T (wie soeben beschrieben) durch ein „Minimum Weight Perfect Matching" für seine ungeraden Knoten zu einem Euler'schen Graphen T'.

3. Konstruiere eine Euler-Tour durch T' und gib diese als Rundreise aus.

Satz 4.4 *Die vom Algorithmus von Christofides konstruierte Approximationsalgorithmus für TSP hat die Güte* 1.5.

Beweis Es sei R_* eine optimale Rundreise mit Kosten $c(R_*)$. Wir wissen bereits, dass $c(R_*) \geq c(T)$. Der Algorithmus von Christofides konstruiert eine Rundreise R mit Kosten $c(T)+c(M)$, wobei M das „Minimum Weight Perfect Matching" bezeichnet und $c(M)$ das Gesamtgewicht der dabei beteiligten Kanten. Die Güte 1.5 ergibt sich, wenn wir $c(R_*) \geq c(M)/2$ nachweisen können. Zu diesem Zweck sei R'_* sie Subtour von R, die resultiert, wenn wir Knoten geraden Grades in R_* auslassen. R_* enthält $2k$ Kanten, sagen wir $e_1, \ldots, e_{2k}$. Offensichtlich bildet sowohl $\{e_1, e_3, \ldots, e_{2k-1}\}$ als auch $\{e_2, e_4, \ldots, e_{2k}\}$ ein perfektes Heiratssystem für die Knoten $u_1, u_2, \ldots, u_{2k}$. Somit gilt

$$c(R_*) \geq c(R'_*) \geq 2c(M) \ ,$$

was den Beweis abschließt. **qed.**

4.2 Approximierbarkeit von KNAPSACK

Sahni hat 1975 folgende Approximationsalgorithmen A_k für KNAPSACK (Eingabe: Gewichte $w_1, \ldots, w_n$, Profite $p_1, \ldots, p_n$ und Gewichtsschranke W) vorgeschlagen:

1. Sortiere die n Objekte nach ihrer „Profitrate", so dass $p_1/w_1 \geq \cdots \geq p_n/w_n$.

2. Zu jeder Teilmenge $I \subseteq \{1, \ldots, n\}$ mit $|I| \leq k$ bestimme ihr Gesamtgewicht $W(I) = \sum_{i \in I} w_i$ und den durch sie realisierten Profit $P(I) = \sum_{i \in I} p_i$.

3. Bestimme einen „Kandidatenrucksack" $R(I)$, in den zunächst die durch I bestimmten Objekte aufgenommen werde, um ihn danach (unter Beachtung der Gewichtsschranke) mit anderen Objekten aufzufüllen. Beim Auffüllen erhalten Objekte mit höherer Profitrate höhere Priorität.

4. Wähle schließlich den profitabelsten Kandidatenrucksack.

Beispiel 4.5 *Wir betrachten eine Eingabe für KNAPSACK mit* $n = 8$ *Objekten und Gewichtsschranke* $W = 110$*. Sowohl die weiteren Eingabeparameter als auch der Beispiellauf von* A_0 *sind aus folgender Tabelle ersichtlich:*

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
p_i	*11*	*21*	*31*	*33*	*43*	*53*	*55*	*65*
w_i	*1*	*11*	*21*	*23*	*33*	*43*	*45*	*55*
b_i	*1*	*1*	*1*	*1*	*1*	*0*	*0*	*0*
W_Σ	*1*	*12*	*33*	*56*	*89*	*89*	*89*	*89*
P_Σ								*139*

Hierbei ist folgendes zu beachten:

- *Die Eingabeparameter sind bereits nach absteigender Profitrate sortiert.*
- b_i *bezeichnet ein Indikatorbit, welches mit dem Wert* 1 *anzeigt, dass Objekt* i *in den Rucksack gesteckt wurde.*
- W_Σ *ist eine dynamische Variable, die on-line das bisher akkumulierte Gewicht mitzählt. Mit Hilfe von* W_Σ *kann entschieden werden, ob das nächste inspizierte Objekt ohne Überschreitung der Gewichtsschranke* $W = 110$ *in den Rucksack gesteckt werden kann.*
- P_Σ *bezeichnet in entsprechender Weise den akkumulierten Profit. Da uns hier nur der endgültige Wert interessiert, haben wir die Zwischenergebnisse nicht angegeben.*

Wir halten als Ergebnis fest, dass A_0 *die Lösung* $I_0 = \{1, 2, 3, 4, 5\}$ *mit Profit* 139 *(und Gewicht* 89*) berechnet.*
Betrachten wir nun einen Beispiellauf von A_1*. Es sei daran erinnert, dass* I *mit* $|I| = k$ *die Menge der Objekte (bestehend aus 1 Objekt im Falle* $k = 1$*) bezeichnet, die vorab in den Rucksack gesteckt werden. Da* A_0 *die Objekte* $1, 2, 3, 4, 5$ *ausgewählt hat, würden die Beispielläufe mit* $I = \{1\}, \{2\}, \{3\}, \{4\}, \{5\}$ *nur das Ergebnis* $I_0 = \{1, 2, 3, 4, 5\}$ *reproduzieren. Wir können uns also auf die Kandatenrucksäcke* $R(I)$ *mit* $I = \{6\}, \{7\}, \{8\}$ *beschränken. Diese drei Beispielläufe sind in den folgenden Tabellen zu sehen (wobei die Eintragungen für das vorab in* $R(I)$ *aufgenommene Objekt zur besseren Kenntlichkeit* **fett** *gedruckt sind):*

i	**6**	*1*	*2*	*3*	*4*	*5*	*7*	*8*
p_i	**53**	*11*	*21*	*31*	*33*	*43*	*55*	*65*
w_i	**43**	*1*	*11*	*21*	*23*	*33*	*45*	*55*
b_i	**1**	*1*	*1*	*1*	*1*	*0*	*0*	*0*
W_Σ	**43**	*44*	*55*	*76*	*99*	*99*	*99*	*99*
P_Σ								*149*

i	**7**	*1*	*2*	*3*	*4*	*5*	*6*	*8*
p_i	**55**	*11*	*21*	*31*	*33*	*43*	*53*	*55*
w_i	**45**	*1*	*11*	*21*	*23*	*33*	*43*	*55*
b_i	**1**	*1*	*1*	*1*	*1*	*0*	*0*	*0*
W_Σ	**45**	*46*	*57*	*78*	*101*	*101*	*101*	*101*
P_Σ								*151*

i	**8**	*1*	*2*	*3*	*4*	*5*	*6*	*8*
p_i	**65**	*11*	*21*	*31*	*33*	*43*	*53*	*55*
w_i	**55**	*1*	*11*	*21*	*23*	*33*	*43*	*45*
b_i	**1**	*1*	*1*	*1*	*0*	*0*	*0*	*0*
W_Σ	**55**	*66*	*67*	*88*	*88*	*88*	*88*	*88*
P_Σ								*118*

Der beste Kandidatenrucksack (und somit die Ausgabe von A_1) ist $R(\{7\})$. Er enthält die Objekte $1, 2, 3, 4, 7$ und erzielt einen Profit von 151 (bei einem Gewicht von 101). Man überlegt sich leicht, dass die optimale Lösung die Objekte $1, 2, 3, 5, 6$ in den Rucksack steckt. Sie erzielt einen Profit von 159 (bei einem Gewicht von 109). Algorithmus A_2 hätte die optimale Bepackung des Rucksackes beim Beispiellauf mit $I = \{5, 6\}$ aufgespürt.

Der Algorithmus A_0 geht im Prinzip nur Profitraten-basiert vor (da er vorab in den Rucksack nur „die leere Menge packt“ und ihn danach Profitraten-basiert auffüllt). Anhand von „teuflisch“ ausgewählten Eingaben **lässt sich zeigen**, dass A_0 keine konstante Güte c mit $c < \infty$ besitzt. Wenn allerdings „kleine Engelchen“ die Eingabe auswählen, dann ist A_0 gar nicht so übel: **Übg.**

Lemma 4.6 *Die Objekte $1, \ldots, n$ seien absteigend nach Profitrate sortiert. Falls die Eingabe die Bedingung*

$$\exists j \in \{1, \ldots, n\} : \sum_{i=1}^{j} w_i = W \tag{1}$$

erfüllt, dann packt A_0 einen optimalen Rucksack.

Beweis Um den Beweis anschaulich zu machen, fassen wir w_i als Preis auf, zu dem (durch Packen des Objektes i in den Rucksack) ein Nutzen von p_i erzielt werden kann. Man bekommt eine „Nutzeneinheit“ umso billiger, je höher die Profitrate p_i/w_i ist. Diese Logik kann nur dadurch gestört werden, dass irgendwann ein Objekt wegen Überschreitung der Gewichtsschranke W nicht mehr in den Rucksack gepackt werden kann, obwohl die Kapazität des Rucksacks noch nicht voll ausgelastet war. Die Bedingung (1) garantiert jedoch, dass diese „Störung“ nicht eintritt. Daher ist $\sum_{i=1}^{j} p_i$ der größtmögliche Gesamtnutzen, der zu einem Preis von $\sum_{i=1}^{j} w_i = W$ erzielt werden kann. **qed.**

Approximationsalgorithmus A_k mit $k \geq 1$ kommt auch mit Eingaben ganz gut zurecht, die sich ein „Teufel“ ausgedacht hat:

Satz 4.7 *A_k ist ein Approximationsalgorithmus für KNAPSACK mit Güte $1 + 1/k$.*

Beweis Es bezeichne I_* die Indexmenge eines profitabelsten Rucksackes R_* mit Profit $P_* = \sum_{i \in I_*} p_i$. Falls $|I_*| \leq k$, dann wäre R_* einer der Kandidatenrucksäcke von A_k. In diesem Fall würde A_k den maximalen Profit erzielen. Für die weitere Diskussion können wir uns also auf den Fall $|I_*| \geq k + 1$ konzentrieren. Wir indizieren die Objekte so um, dass $I_* = \{1, \ldots, k, k + 1, \ldots, k + l\}$, wobei die Objekte $1, \ldots, k$ die k profitabelsten Objekte in R_* seien. Weiterhin seien die Objekte $k + 1, \ldots, k + l$ absteigend nach ihrer Profitrate geordnet. Da $1, \ldots, k$ die k profitabelsten Objekte in R_* sind und da auch $k+1, \ldots, k+l$ zu R_* gehören, folgt

$$\forall \lambda = 1, \ldots, l : \; p_{k+\lambda} \leq \frac{1}{k+1} \left(p_{k+\lambda} + \sum_{i=1}^{k} p_i \right) \leq \frac{P_*}{k+1} \; . \tag{2}$$

Wir betrachten nun den Kandidatenrucksack $R(I)$ mit $I = \{1, \ldots, k\}$. Dieser Rucksack enthält von vorne herein die Objekte $1, \ldots, k$ mit Gesamtgewicht $W_0 := \sum_{i=1}^{k} w_i$ und Gesamtprofit $P_0 = \sum_{i=1}^{k} p_i$. Beachte, dass A_k auf den restlichen Objekten $R := \{k + 1, \ldots, n\}$ bezüglich Gewichtsschranke $W_R = W - W_0$ Profitraten-basiert vorgeht so wie Algorithmus A_0. Es bezeichne $k + \lambda' \in \{k + 1, \ldots, k + l\}$ den kleinsten Index eines nicht in $R(I)$ aufgenommenen Objektes (den es geben muss, wenn $R(I)$ und R_* nicht übereinstimmen, was wir oBdA annehmen). Weiterhin bezeichne $I' \supseteq \{k + 1, \ldots, k + \lambda' - 1\}$ die Menge der Objekte aus R, die sich zum Zeitpunkt der Inspektion von Objekt $k + \lambda'$ in $R(I)$ befinden, so dass A_k mindestens den Profit $P' := P_0 + \sum_{i \in I'} p_i$ erzielt. Da Objekt $k + \lambda'$ nicht in $R(I)$ aufgenommen wurde, muss

$$W' := w_{k+\lambda'} + \sum_{i \in I'} w_i > W_R$$

gelten. Bezüglich Gewichtsschranke W' an Stelle von W_R wäre folgendes geschehen:

- A_k hätte Objekt $k+\lambda'$ ebenfalls aufgenommen (und damit Gewichtsschranke W' genau erreicht).
- Gemäß Lemma 4.6 wäre $I' \cup \{k + \lambda\}$ eine optimale Lösung auf der Objektmenge $I' \cup \{k + \lambda, \ldots, k + l\}$ mit Gewichtsschranke W' gewesen.

Hieraus folgt nun, dass

$$P_* \leq P' + p_{k+\lambda} \overset{(2)}{\leq} P' + \frac{P_*}{k+1} \; ,$$

woraus sich mit einer leichten Rechnung $P_*/P' \leq 1 + 1/k$ ergibt. Da der von $R(I)$ erzielte Profit mindestens P' beträgt, ist der Beweis jetzt abgeschlossen. **qed.**

Anhand von „teuflisch“ ausgewählten Eingaben für KNAPSACK **lässt sich zeigen**, dass A_k mit $k \geq 1$ keine konstante Güte c mit $c < 1 + 1/k$ besitzt. **Übg.**

Der Algorithmus A_k ist für jede Konstante k polynomiell zeitbeschränkt. Wie aber ist die Abhängigkeit der Zeitschranke von dem Güteparameter k?[6] Die Antwort ist etwas frustrierend: allein schon die Anzahl der Kandidatenrucksäcke (sprich: die Anzahl aller Teilmengen $I \subseteq \{1, \ldots, n\}$ mit $|I| \leq k$) ist proportional zu n^k. Die Laufzeit wächst also exponentiell mit k. Das Approximationsschema von Sahni ist daher nur für kleine Werte von k praktikabel.

Definition 4.8 *Ein Algorithmus A für ein Optimierungsproblem Π, der neben der eigentlichen Eingabe einen zusätzlichen Eingabeparameter k erhält und für festes k einen Approximationsalgorithmus A_k der Güte $1 + 1/k$ für Π repräsentiert, heißt* volles Approximationsschema*, wenn sich seine Laufzeit nach oben durch ein bivariates Polynom in der Eingabelänge N und in k beschränken lässt.*

Ibarra und Kim haben 1975 mit einer Technik namens „Rounding and Scaling“ ein volles Approximationsschema für KNAPSACK entworfen. Wir skizzieren im Folgenden die Grundidee hiervon. Es bezeichne $P_n = \sum_{i=1}^{n} p_i$ die Summe aller Einzelprofite, P_* den vom optimalen Rucksack erzielten Profit, $p_{max} := \max_{i=1,\ldots,n} p_i$ den maximalen Profit-Zahlparameter und N die Kodierungslänge der Eingabe E. Wir nehmen im Folgenden oBdA an, dass kein Objekt ein W überschreitendes Gewicht hat. Folglich gilt $P_* \geq p_{max}$. Zweifellos gibt es einen pseudopolynomiellen Algorithmus A, der eine zweidimensionale Tabelle $T = (T[i,j])_{1 \leq i \leq n, 1 \leq j \leq P_n}$ ausfüllt, so dass $T[i,j]$ das minimale Gewicht ist, mit welchem sich ein Profit von mindestens j durch eine geeignete Auswahl aus den Objekten $1, \ldots, i$ erzielen lässt (bzw. $T[i,j] = \infty$, falls $p_1 + \cdots p_i < j$). Eine geeignete Implementierung dieses Algorithmus (dynamisches Programmieren) hat eine Laufzeit, welche polynomiell von N und in p_{max} abhängt. Aus der n-ten Zeile von T kann man leicht den maximal möglichen Profit P_* ablesen. Algorithmus A ist leider nur pseudopolynomiell, weil p_{max} exponentiell groß in Abhängigkeit von N sein kann. Nun ist der Moment gekommen, in dem „Rounding and Scaling“ ins Spiel kommt. Wir skalieren die Eingabe E (mit eventuell riesenhaftem p_{max}) um den Faktor

$$K := \frac{p_{max}}{(k+1)n} \leq \frac{P_*}{(k+1)n} \tag{3}$$

herunter, indem wir die Parameter p_i durch

$$p_i' := \left\lfloor \frac{p_i}{K} \right\rfloor \leq \left\lfloor \frac{p_{max}}{K} \right\rfloor \leq (k+1)n \tag{4}$$

ersetzen.[7] Wir bezeichnen die neue Eingabe mit E'. Der Approximationsalgorithmus von Ibarra und Kim, im folgenden mit A bezeichnet, geht vor wie folgt:

1. Berechne aus Eingabe E mit den Parametern $p_1, \ldots, p_n; w_1, \ldots, w_n; W$ und dem zusätzlichen Eingabeparameter k die Eingabe E' mit den gemäß (3),(4) berechneten Parametern $p_1', \ldots, p_n'$ anstelle von $p_1, \ldots, p_n$.

[6] Genau genommen sollten wir hier einen uniformen Algorithmus A betrachten, der k als zusätzlichen Eingabeparameter erhält und dann vorgeht wie A_k.

[7] Grundsätzlich gilt, dass hohe Werte von K die Laufzeit verbessern, aber die Güte verschlechtern (und umgekehrt für niedrige Werte von K). Die Wahl von K ist somit ein Balance-Akt.

2. Wende den pseudopolynomiellen Algorithmus A' auf E' an und lies aus der resultierenden Tabelle T eine optimale Lösung $I \subseteq \{1, \ldots, n\}$ von E' ab.

3. Gib I als Lösung für die ursprüngliche Eingabe E aus.

Satz 4.9 *Der Algorithmus A von Ibarra und Kim ist ein volles Approximationsschema für KNAPSACK.*

Beweis Wir beginnen mit der Zeitanalyse. Die Laufzeit wird dominiert durch die Berechnung der optimalen Lösung I für die Eingabe E' unter Anwendung des pseudopolynomiellen Algorithmus' A'. Dies kostet größenordnungsmäßig $\text{poly}(N, p_{max}/K) \leq \text{poly}(N, (k+1)n)$ viele Schritte, woraus sich wegen $n \leq N$ eine in N und k polynomiell beschränkte Rechenzeit ergibt.
Abschließend berechnen wir den von I erzielten Profit und vergleichen ihn mit P_*. Bezogen auf Eingabe E bzw. E' erzielt I den Profit

$$P_E = \sum_{i \in I} p_i \text{ bzw. } P_{E'} = \sum_{i \in I} p'_i = \sum_{i \in I} \left\lfloor \frac{p_i}{K} \right\rfloor \leq \frac{P_E}{K} \ .$$

Es ist nicht schwer **zu zeigen**, dass $P_* \leq KP_{E'} + Kn$. Offensichtlich unterhalten dann P_E, P_* und $P_{E'}$ die Beziehung **Übg.**

$$KP_{E'} \leq P_E \leq P_* \leq KP_{E'} + Kn \leq P_E + Kn \ . \tag{5}$$

Nun ergibt sich die Güte $1 + 1/k$ durch eine einfache Rechnung aus

$$P_E \geq P_* - Kn = P_* - \frac{p_{max}}{k+1} \geq P_* - \frac{P_*}{k+1} \ .$$

qed.

4.3 NP-harte Approximation

Einige Optimierungsprobleme (wie zum Beispiel KNAPSACK) erlauben (volle) Approximationsschemata, andere (wie zum Beispiel TSP) besitzen nicht einmal konstante Gütegarantien (sofern $P \neq NP$). Wo liegen die theoretischen Grenzen für die Güte von Approximationsalgorithmen für ein gegebenes Optimierungsproblem? In diesem Abschnitt gehen wir dieser Frage nach und lernen ein paar Techniken zum Nachweis der Nichtapproximierbarkeit kennen.

Als erste Basistechnik zum Nachweis der Nichtapproximierbarkeit eines Optimierungsproblems besprechen wir die Herstellung einer „mupltiplikativen Lücke (multiplicative gap)“ in Bezug auf den zu optimierenden Parameter. In Verbindung mit TSP hatten wir diese Technik implizit bereits kennen gelernt: falls $P \neq NP$, dann besitzt TSP keine konstante Gütegarantie. Wie hatten wir dieses Resultat erzielt? Wesentlich bei der verwendeten Reduktion von HC auf TSP war die folgende Eigenschaft:

Wenn der gegebene Graph $G = (V, E)$ einen Hamilton'schen Kreis besitzt, dann erlaubt die von G induzierte Distanzmatrix eine Rundreise der Länge $n = |V|$. Gibt es hingegen keinen Hamilton'schen Kreis, dann hat die kürzeste Rundreise mindestens die Länge $kn + 1$.
Die Lücke zwischen den Kostenparametern n und $kn + 1$ erlaubt den Schluss:
Wenn ein Algorithmus für TSP die Güte k besitzt, dann kann er (ohne wesentlichen Effizienzverlust) dazu benutzt werden, HC exakt zu lösen.
Die meisten Nichtapproximierbarkeitsresultate beruhen auf einer polynomiellen Reduktion, welche eine multiplikative Lücke (im eben beschriebenen Sinn) herstellt. Einfache Beispiele hierfür ergeben sich durch Optimierungsprobleme, die bereits für eine konstante Kosten- bzw. Profitschranke NP-hart sind. Diese erlauben die Herstellung einer multiplikativen Lücke auf triviale Weise. Wir demonstrieren dies anhand von COLORABILITY:

Bemerkung 4.10 *Falls $P \neq NP$, dann kann kein Approximationsalgorithmus für COLORABILITY eine konstante Güte unterhalb von $4/3$ besitzen.*

Beweis Das Graphenfärbungsproblem ist bereits für Kostenschranke 3 (3-COLORABILITY) NP-vollständig. Ein Approximationsalgorithmus mit Güte $c < 4/3$ kann benutzt werden, um die Frage der 3-Färbbarkeit eines gegebenen Graphen exakt zu lösen. Falls ein solcher Algorithmus existierte, wäre $P = NP$. **qed.**

Völlig analog ergibt sich der

Satz 4.11 *Unter der Voraussetzung $P \neq NP$ gilt folgendes. Wenn ein Minimierungspoblem (bzw. Maximierungsproblem) Π bereits mit konstanter Kostenschranke k NP-hart ist, dann kann kein Approximationsalgorithmus für Π eine konstante Güte unterhalb von $(k + 1)/k$ (bzw. $k/(k - 1)$) besitzen.*

Als zweite Basistechnik verwenden wir den Nachweis der starken NP-Härte. Es gelten nämlich folgende Resultate:

Satz 4.12 *Es sei Π ein Optimierungsproblem mit einem natürlich-zahligen Optimierungsparameter und der Eigenschaft, dass der Wert $C_*(E)$ einer optimalen Lösung für Eingabe E polynomiell in der Kodierungslänge $N(E)$ und in dem maximalen in E vorkommenden Zahlparameter $M(E)$ nach oben beschränkt sind. Dann kann ein volles Approximationsschema für Π in einen pseudopolynomiellen Algorithmus für Π transformiert werden.*

Beweis Wir beschränken uns auf die Betrachtung von einem Minimierungsproblem Π. **Der Beweis für Maximierungsprobleme lässt sich analog führen.** **Übg.**
Nach Voraussetzung gibt es ein Polynom q mit der Eigenschaft

$$C_*(E) < q(N(E), M(E))$$

für alle Eingabeinstanzen E von Π und ein volles Approximationsschema A für Π. Der korrespondierende pseudopolynomielle Algorithmus A' (zur exakten Lösung von Π) geht auf Eingabe E vor wie folgt:

1. $k := q(N(E), M(E))$.
2. Wende A auf Eingabe E und ein $k \geq 1$ an und erhalte eine Lösung mit Profit $C(E) \leq (1 + 1/k)C_*(E)$.

Da die Laufzeit von A polynomiell in $N(E)$ und k beschränkt ist, ist die Laufzeit von A' polynomiell in $N(E)$ und $M(E)$ beschränkt (pseudopolynomielle Laufzeit). Offensichtlich gilt

$$C(E) - C_*(E) \leq \frac{C_*(E)}{k} = \frac{C_*(E)}{q(N(E), M(E))} < 1 \ .$$

Aus $C(E), C_*(E) \in \mathbb{N}_0$ folgt $C(E) = C_*(E)$. **qed.**

Folgerung 4.13 *Falls $P \neq NP$, dann kann ein stark NP-hartes Optimierungsproblem (mit polynomiell in $N(E)$ und $M(E)$ beschränktem Wert einer optimalen Lösung) kein volles Approximationsschema besitzen.*

Gemäß Satz 4.12 lässt sich ein volles Approximationsschema in einen pseudopolynomiellen Algorithmus transformieren. Zumindest für den Spezialfall von KNAPSACK haben Ibarra und Kim die Umkehrung dieses Satzes demonstriert, indem sie (mit der Technik des „Rounding and Scaling“) einen pseudopolynomiellen Algorithmus in ein volles Approximationsschema transformiert haben. Obschon es kein „Metatheorem“ gibt, welches die Konstruktion von Ibarra und Kim auf beliebige pseudopolynomiell lösbare Optimierungsprobleme verallgemeinert, ist die Technik des „Rounding and Scaling“ in vielen Fällen (ähnlich wie bei KNAPSACK) erfolgreich anwendbar.

Approximation bis auf eine „additive Konstante“? Wir haben Approximatiosnalgorithmen kennen gelernt, die das Optimum bis auf einen konstanten Faktor treffen. Stärker wären freilich mathematische Gütegarantien, das Optimum bis auf eine additive Konstante zu treffen. Ist so etwas denkbar? Die Antwort lautet für fast alle natürlichen Optimierungsprobleme NEIN. Wir demonstrieren dies an zwei Beispielen:

Bemerkung 4.14 *Falls $P \neq NP$, dann kann es keinen Approximationsalgorithmus A für KNAPSACK mit einer Gütegarantie von der Form*

$$\exists k \in \mathbb{N}, \forall E : \ C_*(E) - C_A(E) \leq k$$

geben, wobei $C_*(E)$ den auf Eingabe E maximal erzielbaren Profit bezeichnet und $C_A(E)$ den vom Algorithmus A auf E erzielten Profit.

Beweis Angenommen es gäbe einen Approximationsalgorithmus A mit einer solchen Gütegarantie. Dann würde der folgende Algorithmus A' KNAPSACK in Polynomialzeit optimal lösen:

1. Es sei E' die aus E resultierende Eingabe, wenn wir alle Zahlparameter um den Faktor $k+1$ hochskalieren (also mit $k+1$ multiplizieren).

2. Wende A auf E' an und erhalte eine Auswahl I der in den Rucksack gepackten Objekte, deren Gesamtprofit vom maximal erzielbaren Profit additiv nur um maximal k abweicht.

3. Gib I als Lösung für die ursprüngliche Eingabe E aus.

Beachte, dass die Probleme E und E' „isomorph“ sind: Korrespondierende Eingabeparameter und Gesamtprofite von korrespondierenden Lösungen unterscheiden sich immer nur um den Faktor $k+1$. Eine optimale Lösung für E' ist demnach auch eine optimale Lösung für E. Da in E' nur Zahlen auftauchen, die Vielfache von $k+1$ sind, ist „vom Profit einer optimalen Lösung additiv um maximal k abweichen" gleichbedeutend mit „einen optimalen Profit erzielen“. Somit ist A' ein Optimierungsalgorithmus für KNAPSACK (der offensichtlich in Polynomialzeit arbeitet) und es folgt $P = NP$. **qed.**

Der Beweis war darum so leicht, weil ein Zahlenproblem vorlag und wir alle Zahlparameter einfach mit einer geeigneten Konstante multiplizieren konnten. Wie sieht es aber bei rein kombinatorischen Problemen aus? Die Antwort lautet GENAUSO:

Bemerkung 4.15 *Falls $P \neq NP$, dann kann es keinen Approximationsalgorithmus A für Independent Set mit einer Gütegarantie von der Form*

$$\exists k \in \mathbb{N}, \forall E: \ P_*(G) - P_A(G) \leq k$$

geben, wobei $P_*(G)$ die maximale Anzahl paarweise unabhängiger Knoten im Eingabegraphen G bezeichnet und $P_A(G)$ die Mächtigkeit der von Algorithmus A konstruierten unabhängigen Menge in G.

Beweis Gehe vor wie bei KNAPSACK, außer dass die Transformation von G in G' die Technik der „kombinatorischen Multiplikation“ verwendet: G' besteht aus $k+1$ disjunkten Kopien von G. Die weitere Argumentation kann aufgebaut werden wie bei dem entsprechenden Nachweis für KNAPSACK. **qed.**

Da fast alle natürlichen Optimierungsprobleme die „kombinatorische Multiplikation“ der Eingabeinstanz mit einer Konstanten (sprich: die Vervielfältigung der Eingabeinstanz) erlauben, sind „additive Gütegarantien“ im Prinzip nicht erreichbar.

4.4 Komplexitätsklassen für Optimierungsprobleme

Es bezeichne NPO die Klasse aller Optimierungsprobleme, die durch eine polynomiell verifizierbare Relation und eine polynomiell auswertbare Wertefunktion gegeben sind. APX bezeichne die Probleme aus NPO, die einen Approximationsalgorithmus mit konstanter Güte besitzen (die sogenannten „approximierbaren“ Probleme). PTAS bezeichne die Probleme aus

NPO, die ein Approximationsschema besitzen. Schließlich bezeichne FPTAS die Probleme aus NPO, die ein volles Approximationsschema besitzen (wie zum Beispiel KNAPSACK). Es ergibt sich die Hierarchie

$$FPTAS \subseteq PTAS \subseteq APX \subseteq NPO \ ,$$

die unter der Voraussetzung $P \neq NP$ echt ist:

1. „Independent Set“ eingeschränkt auf planare Graphen gehört zu PTAS (ohne Beweis). Da es sich hier nicht um ein Zahlenproblem und daher automatisch um ein stark NP-vollständiges Problem handelt, kann es aber gemäß Folgerung 4.13 kein volles Approximationsschema geben.

2. Probleme wie „Bin Packing“, „Vertex Cover“, und „Metrisches TSP“ besitzen Approximationsalgorithmen mit konstanter Güte, aber kein Approximationsschema (was mit Hilfe des berühmten PCP-Theorems gezeigt werden kann).

3. Probleme wie „Set Cover“, „Graphenfärbung“ und CLIQUE besitzen keine Approximationsalgorithmen mit einer konstanten Güte (was ebenfalls mit dem PCP-Theorem gezeigt werden kann).

Das angesprochene PCP-Theorem und seine Bedeutung für die Theorie der Approximationsalgorithmen werden wir evtl. zu einem späteren Zeitpunkt der Vorlesung besprechen.

Für die Klassen NPO und APX lassen sich mit Hilfe eines geeigneten Reduktionsbegriffes vollständige Probleme (also schwerste Vertreter ihrer Klasse vom Standpunkt der Approximierbarkeit) ausfindig machen. Die uns bekannten Reduktionsbegriffe sind hierfür nicht geeignet. Man braucht vielmehr eine Art „approximationserhaltende“ Reduktion. Populäre approximationserhaltende Reduktionen sind die sogenannten AP- bzw. L-Reduktionen. Wir werden darauf evtl. in einem späteren Stadium der Vorlesung nochmals zurückkommen. Ansonsten sei auf die Webseite

`www.nada.kth.se/theory/compendium/`

verwiesen, die von Viggo Kann gepflegt wird. Sie enthält zu einer großen Liste von grundlegenden Optimierungsproblemen die neuesten „guten“ und „schlechten“ Nachrichten (sprich: verbesserte Approximationsalgorithmen bzw. verbesserte Nachweise der inhärenten Nichtapproximierbarkeit). In einigen Fällen konnte die magische Schwelle k_0 für die bestmögliche Güte exakt bestimmt werden.