Algorithmen 2

Kapitel: Approximationsalgorithmen

Thomas Worsch

Fakultät für Informatik
Karlsruher Institut für Technologie

Wintersemester 2017/2018

Überblick

Suchprobleme

- für jede Probleminstanz $I \in M_I$ gibt es Menge $M_S(I)$ möglicher Lösungen $S \in M_S(I)$
- *Zielfunktion* $f : \bigcup_{I \in M_I} M_S(I) \to \mathbb{R}_+$
- *Minimierungsproblem:* für jedes $I \in M_I$ existiere
  - $f^*(I) = \min\{f(S) \mid S \in M_S(I)\}$
  - gegeben: I
  - gesucht: S mit $f(S) = f^*(I)$
- *Maximierungspoblem:* analog

Approximation bei Suchproblemen

- Lösungen S mit $f(S) = f^*(I)$ sind oft schwer zu finden
  - z. B. **NP**-schwer

Auswege:

- naiv: alle möglichen Lösungen betrachten
  - oft zu aufwändig
- ad-hoc Heuristiken
  - Qualität der Antwort eventuell unklar
- *Approximationsalgorithmus* A
  - $f(A(I))$ «möglichst nahe an» $f^*(I)$
  - *Optimierungsaufgabe*
  - gerne polynomielle Laufzeit
  - mit Garantie für Lösungsgüte

Approximation bei «Zählproblemen»

- für jede Probleminstanz $I \in M_I$ gibt es Menge $M_S(I)$ möglicher Lösungen $S \in M_S(I)$
- *gesucht:* Algorithmus A
  - mit $A(I)$ «möglichst nahe an» $|M_S(I)|$,
  - der die Anzahl der Lösungen annähert
  - Beispiel: bestimme die Anzahl perfekter Matchings eines bipartiten Graphen

- nicht in dieser Vorlesung

Turing-Reduzierbarkeit

Das kam in «Theoretische Grundlagen der Informatik» vor:

- Suchproblem Π **NP**-*schwer* oder **NP**-*hart*,
  falls ein **NP**-vollständiges Entscheidungsproblem L
  existiert mit $L \leq_T^p \Pi$.
- das heißt:
  - es gibt *Orakel-Turingmaschine* für L
  - mit Orakel für Π,
    - jede Befragung des Orakels braucht 1 Schritt
  - die polynomielle Laufzeit hat,
- Turing-Reduktion in Polynomialzeit
- das heißt:
  wenn Π in Polynomialzeit lösbar, dann auch L

Überblick

Job Scheduling: Aufgabenstellung

Problem: *Job Shop Scheduling* oder *Makespan Scheduling*

- *Maschinen* $M_1, \ldots, M_m$
- *Jobs* $J_1, \ldots, J_n$
  - Job J_j benötigt Zeit $t_j \geq 0$
- *Lösung* $S : \{1, \ldots, n\} \longrightarrow \{1, \ldots m\}$
- Last von Maschine i: $L_i = \sum_{S(j)=i} t_j$
- Zielfunktion: minimiere *Makespan* $L_{\max} = \max_i L_i$
  - Wann ist die letzte Maschine fertig?
- einfacher Fall
  - identische Maschinen
  - unabhängige Jobs
- Problem ist **NP**-hart

naheliegender Algorithmus: LISTSCHEDULING

naheliegender Algorithmus: LISTSCHEDULING

```
1  LISTSCHEDULING(n, m, t_{1...n})
2  each L_i ← 0                          ▷ load of machine i, 1 ≤ i ≤ m
3  each S_j ← 0                          ▷ machine for job j, 1 ≤ j ≤ n
4  for each j in range(1, n)
5       do pick k from {i | L_i is currently minimal }
6          S_j ← k
7          L_k ← L_k + t_j
8  return S
```

im folgenden kurz *LS* statt LISTSCHEDULING

Eigenschaften des Algorithmus (1)

Lemma

Es sei $T = \sum_j t_j$. Für jede Lösung S des Algorithmus gilt:
Wenn J_ℓ ein Job ist, der zuletzt fertig wird, dann ist

$$L_{\max} \leq \frac{T}{m} + \frac{m-1}{m} t_\ell$$

Beweis

$t = L_{\max} - t_\ell$ Startzeitpunkt von J_ℓ

Eigenschaften des Algorithmus (1)

Lemma

Es sei $T = \sum_j t_j$. Für jede Lösung S des Algorithmus gilt:
Wenn J_ℓ ein Job ist, der zuletzt fertig wird, dann ist

$$L_{\max} \leq \frac{T}{m} + \frac{m-1}{m} t_\ell$$

Beweis

$t = L_{\max} - t_\ell$ Startzeitpunkt von J_ℓ

- bis hierhin *alle* beschäftigt, mit Jobs $\neq J_\ell$,
  also $mt \leq T - t_\ell$, also

$$L_{\max} = t + t_\ell \leq \frac{T}{m} - \frac{t_\ell}{m} + t_\ell$$

Approximationsfaktor

Approximationsalgorithmus A
für Minimierungsproblem mit Zielfunktion f erzielt
Approximationsfaktor ρ,
falls für alle Probleminstanzen $I \in M_i$ gilt:

$$\frac{f(A(I))}{f^*(I)} \leq \rho$$

$\rho = 1$: A liefert stets eine optimale Lösung

Eigenschaften des Algorithmus (2)

Satz

LS erzielt Aproximationsfaktor $\rho \leq 2 - \frac{1}{m}$.

Beweis

- stets: $f^*(I) \geq T/m$ und für jedes j: $f^*(I) \geq t_j$
- sei I beliebig:

$$\frac{f(LS(I))}{f^*(I)} \leq \frac{T/m}{f^*(I)} + \frac{m-1}{m}\frac{t_\ell}{f^*(I)}$$
$$\leq \frac{T/m}{T/m} + \frac{m-1}{m}\frac{t_\ell}{t_\ell} = 2 - \frac{1}{m}$$

□

Eigenschaften des Algorithmus (3)

Lemma

LS erzielt Aproximationsfaktor $\rho \geq 2 - \frac{1}{m}$.

Beweis

Eigenschaften des Algorithmus (3)

Lemma

LS erzielt Aproximationsfaktor $\rho \geq 2 - \frac{1}{m}$.

Beweis

Probleminstanzen bei denen

- $f(LS(I)) = 2m - 1$ und $f^*(I) = m$:
  - $m(m-1)$ Jobs mit $t_1 = \cdots = t_{n-1} = 1$
  - letzter Job mit $t_n = m$
- LS: alle m Maschinen bis $t = m - 1$ nur Minijobs, dann der große $\quad \rightsquigarrow L_{\max} = m - 1 + m = 2m - 1$
- optimal: eine Maschine nur den großen Job, $m - 1$ machen je m Minijobs $\quad \rightsquigarrow L_{\max} = m$ □

Überblick

Erinnerung: TSP-Suchproblem

- Probleminstanz
  - vollständiger Graph $G = (V, E = V \times V)$ und
  - Längenfunktion $c : E \to \mathbb{Z}_+$
- für Permutation π von V:

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

- $f^*(G, c) = \min_\pi f(\pi)$
- gesucht: Permutation π mit minimalem $f(\pi) = f^*(G, c)$

TSP-a-Approximations-Suchproblem

gegeben: $a \geq 1$

- Probleminstanz
  - vollständiger Graph $G = (V, E = V \times V)$ und
  - Längenfunktion $c : E \longrightarrow \mathbb{Z}_+$
- für Permutation π von V:

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

- gesucht: Permutation π mit $f(\pi) \leq a \cdot f^*(G, c)$

- kurz a-APPROXTSP

Erinnerung: Hamiltonkreis (Entscheidungsproblem)

- Probleminstanz:
  - Graph $G = (V, E)$
- Frage: Gibt es einen Hamiltonkreis in G?
  - i. e. Permutation π so, dass $(\pi(1), \pi(2), \pi(n), \pi(1))$ Kreis

- HAMILTONIANCIRCUIT ist **NP**-vollständig

Schwere Approximierbarkeit des TSP

Satz

Für jede Konstante $a \geq 1$ ist das
TSP-a-Approximations-Suchproblem **NP**-hart.

Beweisidee

Schwere Approximierbarkeit des TSP

Satz

Für jede Konstante $a \geq 1$ ist das
TSP-a-Approximations-Suchproblem **NP**-hart.

Beweisidee

zeige:

- wenn a-APPROXTSP in Polynomialzeit lösbar,
- dann auch HAMILTONIANCIRCUIT

Schwere Approximierbarkeit des TSP: Konstruktion

- gegeben: $G = (V, E)$ mit $n = |V|$
- definiere: $G' = (V, V \times V)$ und Längenfunktion c

Schwere Approximierbarkeit des TSP: Konstruktion

- gegeben: $G = (V, E)$ mit $n = |V|$
- definiere: $G' = (V, V \times V)$ und Längenfunktion c mit

$$c(u, v) = \begin{cases} 1, & \text{falls } (u, v) \in E \\ an + 1, & \text{sonst} \end{cases}$$

- sei AATSP Algorithmus für a-APPROXTSP Suchproblem

Schwere Approximierbarkeit des TSP: Konstruktion

- gegeben: $G = (V, E)$ mit $n = |V|$
- definiere: $G' = (V, V \times V)$ und Längenfunktion c mit

$$c(u, v) = \begin{cases} 1, & \text{falls } (u, v) \in E \\ an + 1, & \text{sonst} \end{cases}$$

- sei AATSP Algorithmus für a-APPROXTSP Suchproblem
- wenn G Hamiltonkreis hat,
  dann hat G' Kreis der Länge n

Schwere Approximierbarkeit des TSP: Konstruktion

- gegeben: $G = (V, E)$ mit $n = |V|$
- definiere: $G' = (V, V \times V)$ und Längenfunktion c mit

$$c(u, v) = \begin{cases} 1, & \text{falls } (u, v) \in E \\ an + 1, & \text{sonst} \end{cases}$$

- sei AATSP Algorithmus für a-APPROXTSP Suchproblem
- wenn G Hamiltonkreis hat,
  dann hat G' Kreis der Länge n
  AATSP(G', c) liefert Lösung $\leq an$

Schwere Approximierbarkeit des TSP: Konstruktion

- gegeben: $G = (V, E)$ mit $n = |V|$
- definiere: $G' = (V, V \times V)$ und Längenfunktion c mit

$$c(u, v) = \begin{cases} 1, & \text{falls } (u, v) \in E \\ an + 1, & \text{sonst} \end{cases}$$

- sei AATSP Algorithmus für a-APPROXTSP Suchproblem
- wenn G Hamiltonkreis hat,
  dann hat G' Kreis der Länge n
  AATSP(G', c) liefert Lösung $\leq an$
- wenn G keinen Hamiltonkreis hat,
  dann hat jeder Kreis von G' Länge $\geq an + 1 + n - 1 > an$

□

Schwere Approximierbarkeit des TSP: Konstruktion

- gegeben: $G = (V, E)$ mit $n = |V|$
- definiere: $G' = (V, V \times V)$ und Längenfunktion c mit

$$c(u, v) = \begin{cases} 1, & \text{falls } (u, v) \in E \\ an + 1, & \text{sonst} \end{cases}$$

- sei AATSP Algorithmus für a-APPROXTSP Suchproblem
- wenn G Hamiltonkreis hat,
  dann hat G' Kreis der Länge n
  AATSP(G', c) liefert Lösung $\leq an$
- wenn G keinen Hamiltonkreis hat,
  dann hat jeder Kreis von G' Länge $\geq an + 1 + n - 1 > an$
  AATSP(G', c) liefert Lösung $> an$

□

Überblick

METRIC TSP

- wie bei TSP, aber zusätzlich wird verlangt, dass c der *Dreiecksungleichung* genügt:
  - für alle $x, y, z \in V$: $c(x, y) + c(y, z) \geq c(x, z)$

- *metrische «Vervollständigung»* $mv(G, c)$ von (G, c):
  $mv(G, c) = (G, c')$ mit

  $$c'(x, y) = \text{ Länge kürzester Wege von } x \text{ nach } y$$

  - «Vervollständigung» hier irreführendes Wort
  - woher kommt das wohl?

- Konstruktion eben:
  Dreiecksungleichung im allgemeinen *verletzt*

Satz

Für Instanzen des METRICTSP Suchproblems kann man in Polynomialzeit eine 2-Approximation berechnen.

Beweisidee

```
1  metricTSPApprox(G, c)
2  T ← MST(G, c)                          ▷ w(T) ≤ f*(G, c)
3  T' ← T with every edge doubled         ▷ w(T') ≤ 2f*(G, c)
4  T'' ← EULERTOUR(T')                    ▷ w(T'') ≤ 2f*(G, c)
5  S ← remove duplicate nodes from T''    ▷ w(S) ≤ 2f*(G, c)
6  return S
```

Überblick

Pseudopolynomielle Laufzeit

Wovon hängt die Laufzeit eines Algorithmus ab?

- Laufzeit $t(I)$ eines Algorithmus für eine Eingabe I abhängig von *Größe* $n(I)$ der Repräsentation von I

üblich:

- $n(I)$ ist Anzahl $n_2(I)$ der Bits um I binär zu codieren
  - Codierung von $k \in \mathbb{N}_+$ braucht $\Theta(\log_2 k)$ Bits
- *polynomielle Laufzeit* $t(n)$:
  exisitert Polynom $p(n)$ für alle I gilt: $t(I) \leq p(n_2(I))$

alternativ:

- $n(I)$ ist Anzahl $n_1(I)$ der Bits um I *unär* zu codieren
  - Codierung von $k \in \mathbb{N}_+$ braucht k Bits
- *pseudopolynomielle Laufzeit* $t(n)$:
  exisitert Polynom $p(n)$ für alle I gilt: $t(I) \leq p(n_1(I))$

Zwei kleine Warnungen

- Algorithmus mit pseudopolynomieller Laufzeit: tatsächliche Form der Eingabe?

- pseudopolynomielle Laufzeit nicht verwechseln mit *quasipolynomieller Laufzeit:*

$$t(n) = 2^{O((\log n)^c)}$$

für eine Konstante $c > 0$

  - Beispiel: $c = 2 \rightsquigarrow n^{\log n}$

KNAPSACK Suchproblem

Probleminstanzen

- endliche Menge $M = \{1, \ldots, n\}$ von *Gegenständen*
- *Maximalgröße* $W \in \mathbb{N}_+$ «Rucksackgröße»
- *Größen* $w_i \in \mathbb{N}_+$ o. B. d. A. jedes $w_i \leq W$
- *Profite* $p_i \in \mathbb{N}_+$

Lösungen

- Teilmenge $M' \subseteq M$ mit
- $w(M') = \sum_{i \in M'} w_i \leq W$

gesucht

- Teilmengen mit möglichst großem Profit
- Zielfunktion $f(M') = p(M') = \sum_{i \in M'} p_i$
- Maximierungsproblem o. B. d. A. $f^*(I) \geq \max p_i$

KNAPSACK: Codierungen der Eingabe

Teil T von I	$u_2(T)$	$u_1(T)$
$\{1, \ldots, n\}$	$\log n$	n
W	$\log W$	W
$\langle w_1, \ldots, w_n \rangle$	$n \log W$	nW
$\langle p_1, \ldots, p_n \rangle$	$n \log \hat{P}$	$n\hat{P}$
insgesamt	$\Theta(n \log W + n \log \hat{P})$	$\Theta(nW + n\hat{P})$

wobei $\hat{P} = \sum_i p_i$

Schwere des KNAPSACK Suchproblems

- **NP**-schwer
  - bei «normaler» Messung der Eingabegröße
  - also Polynomialzeit-Algorithmen unbekannt
- aber: pseudopolynomielle Laufzeit erreichbar
- solche Probleme heißen *schwach* **NP***-schwer*

KNAPSACK: pseudopolynomielle Laufzeit

mit dynamischer Programmierung

- es sei

$$C(i, P) = \min\{w(M') \mid M' \subseteq \{1, \ldots, i\} \wedge p(M') \geq P\}$$

  falls eine solche Teilmenge M' existiert, und ∞ sonst.

- Beobachtung:

$$C(1, P) = \begin{cases} w_1, & \text{falls } p_1 \geq P \\ \infty, & \text{sonst} \end{cases}$$

$$C(i+1, P) = \min(\, C(i, P),\ w_{i+1} + C(i, P - p_{i+1})\,)$$

KNAPSACK: pseudopolynomielle Laufzeit

```
1  dynProgKnapsack(n, W, ⟨w_1, ..., w_n⟩, ⟨p_1, ..., p_n⟩)
2  for P ← 1 to P̂ do
3      C(1, P) ← ...
4  for i ← 1 to n − 1 do
5      for P ← 1 to P̂ do
6          C(i + 1, P) ← min( C(i, P), w_{i+1} + C(i, P − p_{i+1}) )
7  return max{P | C(n, P) ≤ W}
```

Erweiterung

- speichere in $C(i, P)$ auch, welche der Objekte $1, \ldots, i$ benutzt
- Lösung: sogar konkrete Teilmenge/Bitvektor $\mathbf{x}$
- Skalarprodukt $\mathbf{p} \cdot \mathbf{x}$ ist maximaler Profit

Polynomielle Approximationsschemata

Polynomial Time Approximation Scheme (PTAS)

- es sei Π ein $\left\{\begin{matrix}\text{Minimierungs-}\\ \text{Maximierungs-}\end{matrix}\right\}$ Problem
- es sei $\mathcal{A}$ Algorithmus mit Paaren (I, ε) als Eingaben, wobei $\varepsilon > 0$ reell und $I \in \Pi_I$ ist

- $\mathcal{A}$ *polynomielles Approximationsschema (engl. PTAS)*, wenn für jedes $\varepsilon > 0$ ein Polynom $p(n)$ so existiert, dass für jede Instanz I gilt

$$f(\mathcal{A}(I, \varepsilon)) \begin{matrix}\leq\\ \geq\end{matrix} \begin{pmatrix}1+\varepsilon\\ 1-\varepsilon\end{pmatrix} \cdot f^*(I)$$

und Laufzeit $t(I, \varepsilon) \leq p(n_2(I))$

Voll polynomielle Approximationsschemata

Fully Polynomial Time Approximation Scheme (FPTAS)

- es sei Π ein $\left\{\begin{matrix}\text{Minimierungs-}\\ \text{Maximierungs-}\end{matrix}\right\}$ Problem
- es sei $\mathcal{A}$ mit Paaren (I, ε) als Eingaben, wobei $\varepsilon > 0$ reell und $I \in M_I$ ist

- $\mathcal{A}$ *voll polynomielles Approximationsschema (engl. PTAS)*, wenn ein Polynom $p(n, x)$ so existiert, dass für jede Instanz I und jedes $\varepsilon > 0$ gilt

$$f(\mathcal{A}(I, \varepsilon)) \begin{matrix}\leq\\ \geq\end{matrix} \begin{pmatrix}1+\varepsilon\\ 1-\varepsilon\end{pmatrix} \cdot f^*(I)$$

und Laufzeit $t(I, \varepsilon) \leq p(n_2(I), 1/\varepsilon)$

PTAS versus FPTAS

- jedes FPTAS ist ein PTAS
  - $(\frac{n}{\varepsilon})^5 + n^2 + \varepsilon^{-42}$
- aber nicht umgekehrt:
  PTAS- aber nicht FPTAS-Laufzeiten z. B.
  - $n + 2^{1/\varepsilon}$
  - $n^{1/\varepsilon}$
  - $2^{-\log n \log \varepsilon}$

Von pseudopolynomieller zu polynomieller Laufzeit

- $n_2(I) \in \Theta(n \log W + n \log \hat{P})$
- pseudopolynomielle Laufzeit $\in \Theta(n\hat{P})$

Von pseudopolynomieller zu polynomieller Laufzeit

- $n_2(I) \in \Theta(n \log W + n \log \hat{P})$
- pseudopolynomielle Laufzeit $\in \Theta(n\hat{P})$
- polynomielle Laufzeit,
  - wenn die Profite kleine Werte sind,
    z. B. für konstantes k,

$$\hat{P} \in O(n) \cdot 2^k$$

  - z. B. jedes $0 < p_i < 2^k$

FPTAS für KNAPSACK

Schreibweise $\mathbf{p} = \langle p_1, \ldots, p_n \rangle$, etc.

```
1  EPSAPPROXKNAPSACK(ε, n, W, w, p)
2  P ← max_i p_i
3  K ← εP/n
4  each p'_i ← ⌊p_i/K⌋
5  x' ← DYNPROGKNAPSACK(n, W, w, p')
6  return x'
```

FPTAS für KNAPSACK

Schreibweisen

- $\mathbf{x}^*$ optimale Lösung für ursprüngliches $\mathbf{p}$
- $\mathbf{x}'$ optimale Lösung für $\mathbf{p}'$
- dann

$$\mathbf{p} \cdot \mathbf{x}^* = \sum_{i, x_i^* = 1} p_i = \text{ maximaler Profit des Originalproblems}$$

- *Frage:* Wie gut ist $\mathbf{p} \cdot \mathbf{x}'$ im Vergleich zu $\mathbf{p} \cdot \mathbf{x}^*$?

EPSAPPROXKNAPSACK ist FPTAS

Satz

EPSAPPROXKNAPSACK ist ein voll polynomielles Approximationsschema für KNAPSACK.

Lemma

Mit den Bezeichnungen wie bisher

$$\mathbf{p} \cdot \mathbf{x}' \geq (1 - \varepsilon)\mathbf{p} \cdot \mathbf{x}^*$$

EPSAPPROXKNAPSACK ist FPTAS: Approximation

- $K = \varepsilon P/n$ und $p'_i = \lfloor p_i/K \rfloor$, also $Kp'_i \leq p_i$

$$
\begin{aligned}
\mathbf{p} \cdot \mathbf{x}' \geq K\mathbf{p}' \cdot \mathbf{x}' &\geq K\mathbf{p}' \cdot \mathbf{x}^* && \text{weil } \mathbf{x}' \text{ optimal für } \mathbf{p}' \\
&= \sum_{i \in \mathbf{x}^*} K \left\lfloor \frac{p_i}{K} \right\rfloor \\
&\geq \sum_{i \in \mathbf{x}^*} K \left(\frac{p_i}{K} - 1 \right) && = \sum_{i \in \mathbf{x}^*} p_i - \sum_{i \in \mathbf{x}^*} K \\
&\geq \mathbf{p} \cdot \mathbf{x}^* - nK \\
&= \mathbf{p} \cdot \mathbf{x}^* - \varepsilon P \\
&\geq \mathbf{p} \cdot \mathbf{x}^* - \varepsilon \mathbf{p} \cdot \mathbf{x}^* && \text{weil Optimum } \geq \max p_i = P \\
&= (1 - \varepsilon)\mathbf{p} \cdot \mathbf{x}^*
\end{aligned}
$$

EPSAPPROXKNAPSACK ist FPTAS: Polynomialzeit

Lemma

Die Laufzeit von EPSAPPROXKNAPSACK ist in $O(n^3 \cdot \frac{1}{\varepsilon})$.

- dynamischen Programmierung für $\mathbf{p}'$-Problem dominiert: Laufzeit $n\hat{P}'$
- es ist (mit $P = \max_i p_i$)

$$n\hat{P}' = n \sum_i p'_i \leq n^2 \max_i p'_i = n^2 \left\lfloor \frac{P}{K} \right\rfloor = n^2 \left\lfloor \frac{Pn}{\varepsilon P} \right\rfloor \leq n^3 \cdot \frac{1}{\varepsilon}$$

Varianten von Approximation: zwei Beispiele

- bestes bekanntes FPTAS linear in n
- in der Praxis erreicht man «fast Linearzeit»