Algorithmen / Algorithms II

Peter Sanders

Exercise:
Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:
http://algo2.iti.kit.edu/AlgorithmenII_WS20.php
8 Approximation Algorithms

A possibility to **tackle NP-hard problems**

Observation: Almost all interesting optimization problems are NP-hard

Options:

☐ Still try to find an optimal solution but risk that the algorithm doesn’t finish

☐ Ad-hoc heuristics. Will find a solution but how good is it?

☐ **Approximation algorithms:**

  Polynomial running time.
  
  Solutions *guaranteed* to be “close” to optimal.

☐ Redefine/specialize Problem...
Scheduling of independent weighted jobs on parallel machines

$x(j)$: machine that runs Job $j$

$L_i$: $\sum_{x(j)=i} t_j$, Load of machine $i$

Objective function: Minimize makespan

$L_{\text{max}} = \max_i L_i$

Details: identical machines, independent jobs, known running times, offline
List Scheduling

ListScheduling\( (n, m, t) \)

\[ J := \{1, \ldots, n\} \]

array \( L[1..m] = [0, \ldots, 0] \)

\textbf{while} \( J \neq \emptyset \) \textbf{do}

\hspace{1em} pick any \( j \in J \)

\hspace{1em} \( J := J \setminus \{j\} \)

\hspace{1em} // Shortest Queue:

\hspace{1em} pick \( i \) such that \( L[i] \) is minimized

\hspace{1em} \( x(j) := i \)

\hspace{1em} \( L[i] := L[i] + t_j \)

\textbf{return} \( x \)
Many small jobs

**Lemma 1.** If $\ell$ is the job that finishes last, then

$$L_{\text{max}} \leq \sum_j \frac{t_j}{m} + \frac{m-1}{m} t_\ell$$

**Proof**

$$L_{\text{max}} = t + t_\ell \leq \sum_{j \neq \ell} \frac{t_j}{m} + t_\ell = \sum_j \frac{t_j}{m} + \frac{m-1}{m} t_\ell$$

$$= t \cdot m \leq \text{all} - t_1$$
Lower bounds

Lemma 2. $L_{\text{max}} \geq \sum_j \frac{t_j}{m}$

Lemma 3. $L_{\text{max}} \geq \max_j t_j$
The approximation ratio

Definition:

A minimization algorithms achieves approximation ratio $\rho$ with respect to a objective function $f$ if for all inputs $I$, it finds a solution $\mathbf{x}(I)$, such that

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

where $\mathbf{x}^*(I)$ is the optimal solution for input $I$. 
**Theorem:** ListScheduling achieves approximation ratio $2 - \frac{1}{m}$.

**Proof:**

\[
\frac{f(x)}{f(x^*)} \leq \frac{\sum_j t_j/m}{f(x^*)} + \frac{m-1}{m} \cdot \frac{t_\ell}{f(x^*)} \leq 1 + \frac{m-1}{m} \cdot \frac{t_\ell}{f(x^*)} \leq 1 + \frac{m-1}{m} = 2 - \frac{1}{m}
\]

(upper bound Lemma 1)

(lower bound Lemma 2)

(lower bound Lemma 3)
This bound is optimal

Input: \( m(m - 1) \) jobs of size 1 and one job of size \( m \).

List Scheduling: 2m−1  

OPT: m

Therefore, the approximation ratio is \( \geq 2 - 1/m \).
More About Scheduling)

- 4/3 approximation: Sort jobs decreasing by size. Then list scheduling. Time $O(n \log n)$.

- Fast 7/6 approximation: Guess makespan (binary search). then Best Fit Decreasing.

- PTAS ... later ...

- Uniform machines: Machine $i$ has speed $v_i$, job $j$ needs time $t_{ji}/v_i$ on machine $j$. $\Rightarrow$ relatively easy generalization

- Unrelated machines: Job $j$ needs time $t_{ji}$ on machine $j$. 2 approximation. Very different algorithm.

- And many more: different objective functions, order restrictions, ...
Inapproximability of the Traveling Salesman Problem (TSP)

Given a graph $G = (V, V \times V)$, find a simple cycle $C = (v_1, v_2, \ldots, v_n, v_1)$ such that $n = |V|$ and $\sum_{(u,v) \in C} d(u, v)$ is minimized.

**Theorem:** Approximate TSP to any ratio $a$ is NP-hard.

**Proof idea:** It is sufficient to show that $\text{HamiltonCycle} \leq_p a$-approximation of TSP
\(a\)-Approximation of TSP

Given:

Graph \(G = (V, V \times V)\) with edge weights \(d(u, v)\), parameter \(W\).

We need an algorithm with the following properties:

\([G, W]\) is accepted \(\rightarrow \exists\) tour with weight \(\leq aW\).

\([G, W]\) is rejected \(\rightarrow \emptyset\) tour with weight \(\leq W\).
HamiltonCycle $\leq_p$ a Approximation of TSP

Let $G = (V, E)$ an arbitrary undirected graph.

Define $d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 1 + an & \text{else} \end{cases}$

$\exists$ TSP tour with cost $n$

If and only if $G$ has a Hamiltonian cycle

(otherwise: optimal cost $\geq (n - 1) \cdot 1 + (an + 1) = an + n > an$)

Decision algorithms for Hamiltonian cycle:


Is accepted

$\rightarrow \exists$ tour with weight $\leq an$

$\rightarrow \exists$ tour with weight $n \rightarrow \exists$ Hamiltonian path

otherwise $\nexists$ Hamiltonian path
TSP with Triangle Inequality

$G$ (undirected) satisfies triangle inequality

$\forall u, v, w \in V : d(u, w) \leq d(u, v) + d(v, w)$

Metric completion

Consider arbitrary undirected graph $G = (V, E)$ with weight function $c : E \rightarrow \mathbb{R}_+$. Define $d(u, v) := \text{Length of shortest path from } u \text{ to } v$

Example: (undirected) road graph $\rightarrow$ distance table
Eulerian Path/Cycle

Consider arbitrary connected undirected (multi-)graph $G = (V, E)$ with $|E| = m$.

A path $P = \langle e_1, \ldots, e_m \rangle$ is called a Eulerian cycle if $\{e_1, \ldots, e_m\} = E$. (every edge is visited exactly once)

Theorem: $G$ has Eulerian cycle iff $G$ is connected and $\forall v \in V : \text{degree}(v)$ is even.

Eulerian cycles can be found in time $O(|E| + |V|)$. 
2 Approximation by Minimum Spanning Tree

Lemma 4.
Total weight of an MST $\leq$
Total weight of every TSP tour

Algorithm:

$T := \text{MST}(G)$  // weight($T$) $\leq$opt

$T' := T$ with every edge doubled  // weight($T'$) $\leq$2opt

$T'' := \text{EulerianCycle}(T')$  // weight($T''$) $\leq$ 2opt

output removeDuplicates($T''$)  // shortcutting
Example

input weight: 1 2 doubled MST

MST

output

weight 10

Euler cycle

12131415161

optimal weight: 6
Proof of Weight $\text{MST} \leq \text{Weight TSP tour}$

Let $T$ be the optimal TSP tour.
Removing an edge makes $T$ lighter.
Now $T$ is a spanning tree
that cannot be lighter than the MST

General technique: Relaxation

here: a TSP path is a special case of a spanning tree
More TSP

- In practice better 2 approximations, e.g. lightest edge first
- Relatively easy but impractical 3/2 approximation
  (MST + min. weight perfect matching + Eulerian cycle)
- PTAS for Euclidean TSP
- Guinea pig for virtually every optimization heuristic
- Optimal solutions for practical inputs. Rule of thumb:
  If it fits into memory, you can solve it.
  [http://www.tsp.gatech.edu/concorde.html]
  Six-figure number of code lines.
- TSP-like applications are usually more complicated
Pseudo-Polynomial Time Algorithms

$\mathcal{A}$ is pseudo-polynomial time algorithms if

$$\text{Time}_{\mathcal{A}}(n) \in P(n)$$

where $n$ is the number of input bits, if all numbers are in unary coding ($k \equiv 1^k$).
Example: Knapsack Problem

- $n$ items with weight $w_i \in \mathbb{N}$ and value $p_i$.

  Wlog: $\forall i \in 1..n : w_i \leq W$

- Choose a subset $x$ of items

- Such that $\sum_{i \in x} w_i \leq W$ and

- Maximize the value $\sum_{i \in x} p_i$
Dynamic Programming by Value

\[ C(i, P) := \text{smallest capacity for items } 1, \ldots, i \text{ that add up to value } \geq P. \]

Lemma 5.

\[ \forall 1 \leq i \leq n : C(i, P) = \min(C(i - 1, P), \ C(i - 1, P - p_i) + w_i) \]
Dynamic programming by value

Let \( \hat{P} \) be an upper bound for the value (e.g. \( \sum_i p_i \)).

**Time:** \( O(n\hat{P}) \) pseudo-polynomial

  e.g. fill \( 0..n \times 0..\hat{P} \) table \( C(i, P) \) column-wise

**Space:** \( \hat{P} + O(n) \) machine words plus \( \hat{P}n \) bits.
**Fully Polynomial Time Approximation Scheme**

Algorithm $\mathcal{A}$ is a (Fully) Polynomial Time Approximation Scheme for minimization problem $\Pi$ if:

- Input: Instance $I$, error parameter $\varepsilon$
- Output quality: $f(x) \leq (1 + \varepsilon)\text{opt}$
- Time: Polynomial in $|I|$ (and $1/\varepsilon$)
### Examples for bounds

<table>
<thead>
<tr>
<th>PTAS</th>
<th>FPTAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n + 2^{1/\varepsilon}$</td>
<td>$n^2 + \frac{1}{\varepsilon}$</td>
</tr>
<tr>
<td>$n \log \frac{1}{\varepsilon}$</td>
<td>$n + \frac{1}{\varepsilon^4}$</td>
</tr>
<tr>
<td>$\frac{1}{n\varepsilon}$</td>
<td>$n / \varepsilon$</td>
</tr>
<tr>
<td>$n^{42}/\varepsilon^3$</td>
<td>:</td>
</tr>
<tr>
<td>$n + 2^{1000}/\varepsilon$</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
FPTAS for Knapsack

\[ P := \max_i p_i \quad // \text{maximum single value} \]
\[ K := \frac{\varepsilon P}{n} \quad // \text{scaling factor} \]
\[ p'_i := \left\lfloor \frac{p_i}{K} \right\rfloor \quad // \text{scaled values} \]
\[ x' := \text{dynamicProgrammingByProfit}(p', w, C) \]
return \( x' \)
Lemma 6. \( p \cdot x' \geq (1 - \varepsilon)\text{opt}. \)

Proof: Consider the optimal solution \( x^* \).

\[
p \cdot x^* - Kp' \cdot x^* = \sum_{i \in x^*} \left( p_i - K \left\lfloor \frac{p_i}{K} \right\rfloor \right)
\]

\[
\leq \sum_{i \in x^*} \left( p_i - K \left( \frac{p_i}{K} - 1 \right) \right) = |x^*|K \leq nK,
\]

so, \( Kp' \cdot x^* \geq p \cdot x^* - nK \). Also,

\[
Kp' \cdot x^* \leq Kp' \cdot x' = \sum_{i \in x'} K \left\lfloor \frac{p_i}{K} \right\rfloor \leq \sum_{i \in x'} K \frac{p_i}{K} = p \cdot x'. \text{ Thus,}
\]

\[
p \cdot x' \geq Kp' \cdot x^* \geq p \cdot x^* - nK = \text{opt} - \varepsilon P \geq (1 - \varepsilon)\text{opt}
\]

\[\leq \text{opt}\]
Lemma 7. Running time $O(n^3/\varepsilon)$.

Proof. The running time $O\left(n^{\hat{P}'}\right)$ of dynamic programming dominates:

$$n^{\hat{P}'} \leq n \cdot \left(n \cdot \max_{i=1}^{n} p_i'\right) = n^2 \left\lfloor \frac{P}{K} \right\rfloor = n^2 \left\lfloor \frac{Pn}{\varepsilon P} \right\rfloor \leq \frac{n^3}{\varepsilon}.$$
The Best Known FPTAS

\[ O\left( \min \left\{ n \log \frac{1}{\varepsilon} + \frac{\log^2 \frac{1}{\varepsilon}}{\varepsilon^3}, \ldots \right\} \right) \]

- Fewer buckets \( C_j \) (non-uniform)
- Sophisticated dynamic programming

[Kellerer, Pferschy 04]
Optimal Algorithms for the Knapsack Problem

Near linear running time for almost all inputs! In theory and practice.
