Approximation Algorithms

Bertrand Simon (University of Bremen)

June 13, 2019

Polynomial time approximation schemes (PTAS)

The Knapsack problem

The knapsack problem

Input: n items of integer size s_i and integer value v_i
a knapsack of integer size B

Goal: select a subset of items that fits in the knapsack ($\sum s_i \leq B$) and maximizes the value ($\sum v_i$)

Knapsack: first results

Input: n items of size s_i and value v_i, a knapsack of size B

Theorem

The decision version of the knapsack problem is NP-hard.

Knapsack: first results

Input: n items of size s_i and value v_i, a knapsack of size B

Theorem

The decision version of the knapsack problem is NP-hard.

Greedy algorithm:

1. Sort items by decreasing v_i/s_i *(high value, small size first)*
2. $k \leftarrow$ maximum j such that $\sum_{i \leq j} s_i \leq B$
3. Put either the first k items, or the item indexed $k+1$

Lemma (Proof: exercise)

This is a $(1/2)$ *- approximation of the Knapsack problem.*

Knapsack: an exact algorithm

Dynamic Programming

Core idea: rely on optimal solutions for several sub-problems.

Knapsack: an exact algorithm

Dynamic Programming

Core idea: rely on optimal solutions for several sub-problems.

$T[i, v]$: minimum size to achieve value v using items 1 to i

(if value v is not possible, $T[i, v] = \infty$)

How to compute the table T?

- $T[i, v] = \min\{T[i-1, v]; T[i-1, v-v_i] + s_i\}$

Knapsack: an exact algorithm

Dynamic Programming

Core idea: rely on optimal solutions for several sub-problems.

$T[i, v]$: minimum size to achieve value v using items 1 to i

(if value v is not possible, $T[i, v] = \infty$)

How to compute the table T?

- $T[i, v] = \min\{T[i-1, v]; T[i-1, v-v_i] + s_i\}$

The optimal value for the Knapsack problem is:

- $\max\{v \in [0; V] \mid T[n, v] \leq B\}$ with $V = \sum_{i \leq n} v_i$

Knapsack: an exact algorithm

Dynamic Programming

Core idea: rely on optimal solutions for several sub-problems.

$T[i, v]$: minimum size to achieve value v using items 1 to i

(if value v is not possible, $T[i, v] = \infty$)

How to compute the table T?

- $T[i, v] = \min\{T[i-1, v]; T[i-1, v-v_i] + s_i\}$

The optimal value for the Knapsack problem is:

- $\max\{v \in [0; V] \mid T[n, v] \leq B\}$ with $V = \sum_{i \leq n} v_i$

Note: the solution set can be computed backwards from $T[n, v^*]$

Complexity remarks

Input: n items of size s_i and value v_i, a knapsack of size B

Theorem

The Knapsack problem can be solved optimally in time $O(nV)$.

This is a NP-hard problem... Why don't we have $P = NP$?

Complexity remarks

Input: n items of size s_i and value v_i, a knapsack of size B

Theorem

The Knapsack problem can be solved optimally in time $O(nV)$.

This is a NP-hard problem... Why don't we have $P = NP$?

The input size is proportional to $(\log V)$ and not to V
$\Longrightarrow$ The Dynamic Programming algorithm is pseudo-polynomial

Complexity remarks

Input: n items of size s_i and value v_i, a knapsack of size B

Theorem

The Knapsack problem can be solved optimally in time $O(nV)$.

This is a NP-hard problem... Why don't we have $P = NP$?

The input size is proportional to $(\log V)$ and not to V
$\Longrightarrow$ The Dynamic Programming algorithm is pseudo-polynomial

Corollary

The Knapsack problem can be solved optimally in polynomial time if $V = poly(n)$.

Can we use this result to build a polynomial-time approximation algorithm?

Polynomial Time Approximation Schemes

Definition

A *polynomial time approximation scheme* (PTAS) is a family of algorithms A_ε, such that for any $\varepsilon > 0$, algorithm A_ε is a $(1 \pm \varepsilon)$-approximation algorithm that has running time polynomial in the input size for any ε.

Note: the running time may be $O(n^{1/\varepsilon})$.

Polynomial Time Approximation Schemes

Definition

A *polynomial time approximation scheme* (PTAS) is a family of algorithms A_ε, such that for any $\varepsilon > 0$, algorithm A_ε is a $(1 \pm \varepsilon)$-approximation algorithm that has running time polynomial in the input size for any ε.

Note: the running time may be $O(n^{1/\varepsilon})$.

Definition

A *fully polynomial time approximation scheme* (FPTAS) is a PTAS with running time polynomial in both the input size and $1/\varepsilon$.

Example: $O(n^3/\varepsilon^4)$

Towards an FPTAS for Knapsack

Roadmap:

1. Given ε, modify the v_i so that $V = poly(n/\varepsilon)$
2. Solve optimally the modified instance using the DP
3. Check the accuracy (i.e., approximation factor $\geq 1 - \varepsilon$)

Towards an FPTAS for Knapsack

Roadmap:

1. Given ε, modify the v_i so that $V = poly(n/\varepsilon)$
2. Solve optimally the modified instance using the DP
3. Check the accuracy (i.e., approximation factor $\geq 1 - \varepsilon$)

Sketch:

- Given a value μ, round all the v_i to $v'_i = \lfloor v_i/\mu \rfloor$
- Solve this instance optimally

$\Rightarrow$ Actual value of the obtained solution: $SOL \geq OPT - n\mu$.

Towards an FPTAS for Knapsack

Roadmap:

1. Given ε, modify the v_i so that $V = poly(n/\varepsilon)$
2. Solve optimally the modified instance using the DP
3. Check the accuracy (i.e., approximation factor $\geq 1 - \varepsilon$)

Sketch:

- Given a value μ, round all the v_i to $v'_i = \lfloor v_i/\mu \rfloor$
- Solve this instance optimally

$\Rightarrow$ Actual value of the obtained solution: $SOL \geq OPT - n\mu$.

Choose the right μ value: $n\mu \leq \varepsilon OPT \Rightarrow SOL \geq (1-\varepsilon)OPT$

- Define $M := \max_i v_i$ $(\leq OPT)$
- Select $\mu = \varepsilon M/n$
- Verify $V' = \sum_i v'_i \leq \sum_i v_i/\mu \leq nM/\mu \leq n^2/\varepsilon = poly(n, \varepsilon)$

FPTAS for Knapsack

Complete algorithm

1. Let $\mu = \varepsilon M/n$, with $M = \max_i v_i$
2. Let $v'_i = \lfloor v_i/\mu \rfloor$
3. Solve optimally this modified instance using the DP
4. Return the set of items computed by the DP

Complexity: $O(n^3/\varepsilon)$

Correctness proof (S = algorithm set, O = optimal set):

$$\begin{aligned}
\sum_{i\in S} v_i &\geq \sum_{i\in S} \mu v'_i & &\geq \sum_{i\in O} \mu v'_i \\
&\geq \Big(\sum_{i\in O} v_i\Big) - n\mu & &\geq \Big(\sum_{i\in O} v_i\Big) - \varepsilon M \\
&\geq (1-\varepsilon)OPT
\end{aligned}$$

A PTAS for the $P||C_{\max}$ problem

Definition of the problem $P||C_{\max}$

Input: m identical machines,
n jobs, J with processing times $p_1, \ldots, p_n$

Goal: assign each job $j \in J$ to a machine $i \in [m]$
minimizing the maximum load of any machine
$C_{\max} := \max_{i \in [m]} \sum_{j\,:\,j \to i} p_j$

Theorem

$P||C_{\max}$ is NP-hard.

PTAS idea [Hochbaum, Shmoys 87]

- Use a subroutine needing a good makespan estimate
  - Round job sizes to get a bounded number of distinct ones
  - Compute a schedule almost achieving the estimate
- Use binary search to guess the optimal makespan

The PTAS subroutine (1/2)

Theorem

For a given feasible makespan T and any $\varepsilon > 0$, the following algorithm returns a feasible schedule with makespan $\leq (1+\varepsilon)T$.

First step: rounding

Define *Big jobs* as $J_{big} = \{j \in J \mid p_j > \varepsilon T\}$

Round down processing times:
$p_j \in [\varepsilon T(1+\varepsilon)^i, \varepsilon T(1+\varepsilon)^{i+1}) \;\rightarrow\; p'_j = \varepsilon T(1+\varepsilon)^i$

Note: - New values are close to the originals: $p_{j'} \leq p_j \leq (1+\varepsilon)p_{j'}$
- At most $k_\varepsilon := \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil$ distinct new p'_j's

The PTAS subroutine (2/2)

Second step: big jobs (recall only k_ε distinct p'_j in J_{big})

On J_{big}, run the following DP: *(Bin packing problem)*

$A[i_1 \ldots, i_{k_\varepsilon}]$: number of machines needed to schedule i_j jobs of processing time $\varepsilon T(1+\varepsilon)^j$ for each $j \leq k_\varepsilon$ within makespan T

$A[i_1 \ldots, i_{k_\varepsilon}] = \min_{x_1 \ldots, x_{k_\varepsilon}} (A[x_1 \ldots, x_{k_\varepsilon}] + A[i_1 - x_1 \ldots, i_{k_\varepsilon} - x_{k_\varepsilon}])$

Complexity: $O(n^{2k_\varepsilon})$

- If the solution uses more than m machines, output: NO
- Else, save an optimal allocation (actual makespan $\leq T(1+\varepsilon)$)

The PTAS subroutine (2/2)

Second step: big jobs (recall only k_ε distinct p'_j in J_{big})

On J_{big}, run the following DP: *(Bin packing problem)*

$A[i_1 \ldots, i_{k_\varepsilon}]$: number of machines needed to schedule i_j jobs of processing time $\varepsilon T(1+\varepsilon)^j$ for each $j \leq k_\varepsilon$ within makespan T

$A[i_1 \ldots, i_{k_\varepsilon}] = \min_{x_1 \ldots, x_{k_\varepsilon}} (A[x_1 \ldots, x_{k_\varepsilon}] + A[i_1 - x_1 \ldots, i_{k_\varepsilon} - x_{k_\varepsilon}])$

Complexity: $O(n^{2k_\varepsilon})$

- If the solution uses more than m machines, output: NO
- Else, save an optimal allocation (actual makespan $\leq T(1+\varepsilon)$)

Third step: complete the solution

Add small jobs from $J \setminus J_{big}$ with list scheduling.

- If the schedule has makespan $> (1+\varepsilon)T$, output: NO
- Otherwise, output the schedule

A PTAS for $P||C_{\max}$

Conclusion

If we knew $T = C^*_{\max}$, we were done.

A PTAS for $P||C_{\max}$

Conclusion

If we knew $T = C^*_{\max}$, we were done.

Binary search

We find approximately correct $C^*_{\max}$ by binary search.

A PTAS for $P||C_{\max}$

Conclusion

If we knew $T = C^*_{\max}$, we were done.

Binary search

We find approximately correct $C^*_{\max}$ by binary search.

Binary search initialized by:

- LB: $\max\{\sum p_j/m, \max p_j\} \leq C_{\max}$
- UB: list scheduling, so UB $\leq$ 2LB

Loop: Set $T = \frac{\text{UB-LB}}{2}$ and run the previous algorithm
If NO, then LB $:= T$, else UB $:= T$

Check: If UB-LB $\leq \varepsilon$LB, output UB solution.
This takes at most $\log_2 \frac{1}{\varepsilon}$ iterations.

Output makespan:

$$M \leq (1+\varepsilon)T_{last} \leq (1+\varepsilon)(C^*_{\max}+\varepsilon\text{LB}) \leq (1+\varepsilon)^2 C^*_{\max} \leq (1+3\varepsilon)C^*_{\max}$$

A PTAS for $P||C_{\max}$

Running time

The running time of the algorithm is

$$O(\#\text{ iteration of binary search})O(\text{DP time}) = O(\log(1/\varepsilon)n^{2k_\varepsilon}),$$

where $k_\varepsilon = \left\lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \right\rceil$.

Thus, this algorithm is a PTAS.

What you should remember

Definition of a PTAS

- Family of algorithms: $\varepsilon \longrightarrow (1+\varepsilon)$-approximation
- FPTAS: also polynomial in $1/\varepsilon$

Dynamic programming

- Type of algorithm which relies on the optimal solutions of many subproblems

Famous NP-hard problems

- Knapsack: pseudo-polynomial algorithm, FPTAS
- $P||C_{\max}$: PTAS