The Design and Analysis of Algorithms

Lecture 23 Approximation Algorithm IV

Zhenbo Wang

Department of Mathematical Sciences, Tsinghua University

Content

Polynomial-time Approximation Scheme (PTAS)

Def. *P*TAS $(1+\epsilon)$ -approximation algorithm for any constant $\epsilon > 0$.

- Knapsack.
- Load balancing.
- Euclidean TSP.

  S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. *FOCS'96*.

- PTAS produces arbitrarily high quality solution, but trades off accuracy for time.

Knapsack problem

- Given n objects and a knapsack.
- Knapsack has weight limit W.
- Item i has value $v_i > 0$ and weighs $0 < w_j \leq W$.
- Goal: fill knapsack so as to maximize total value.
- Example: $\{3, 4\}$ has value 40.

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

original instance (W = 11)

Knapsack is NP-complete

Knapsack.

Given a set X, weights $w_i \geq 0$, values $v_i \geq 0$, a weight limit W, and a target value V, is there a subset $S \subseteq X$ such that:

$$\begin{aligned} \sum_{i \in S} w_i &\leq W, \\ \sum_{i \in S} v_i &\geq V. \end{aligned}$$

Subset-sum.

Given a set X, values $u_i \geq 0$, and an integer U, is there a subset $S \subseteq X$ whose elements sum to exactly U?

Knapsack is NP-complete

Theorem 1

Subset-sum $\le_P$ Knapsack.

Pf. Given instance $(u_1, \cdots, u_n, U)$ of Subset-sum, construct Knapsack instance:

$$\begin{array}{lccc} v_i = w_i = u_i & \sum_{i \in S} w_i & \le & W, \\ V = W = U & \sum_{i \in S} v_i & \ge & V. \ \square \end{array}$$

Knapsack Problem: Dynamic Programming I

Def. $OPT(i, w)$ = max value subset of items $1, \cdots, i$ with weight limit w.

Case 1. OPT does not select item i.

OPT selects best of $1, \cdots, i-1$ using up to weight limit w.

Case 2. OPT selects item i.

New weight limit $= w - w_i$.

OPT selects best of $1, \cdots, i-1$ using up to weight limit $w - w_i$.

Knapsack Problem: Dynamic Programming I

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0, \\ OPT(i-1,w) & \text{if } w_i > w, \\ \max\{OPT(i-1,w), v_i + OPT(i-1,w-w_i)\} & \text{otherwise .} \end{cases}$$

Theorem 2

Computes the optimal value in $O(nW)$ time.

- Not polynomial in input size.
- Polynomial in input size if weights are small integers.

Knapsack Problem: Dynamic Programming II

Def. $OPT(i, v) =$ min weight of a knapsack for which we can obtain a solution of value $\geq v$ using a subset of items $1, \cdots, i$.

Note. Optimal value is the largest value v such that $OPT(n, v) \leq W$.

Case 1. OPT does not select item i.

OPT selects best of $1, \cdots, i-1$ that achieves value v.

Case 2. OPT selects item i.

Consumes weight w_i, and OPT selects best of $1, \cdots, i-1$ that achieves value $v - v_i$.

$$OPT(i, v) = \begin{cases} 0 & \text{if } v = 0, \\ \infty & \text{if } i = 0, v > 0, \\ \min\{OPT(i-1, v), w_i + OPT(i-1, v - v_i)\} & \text{otherwise} \end{cases}.$$

Knapsack Problem: Dynamic Programming II

Theorem 3

Dynamic programming algorithm II computes the optimal value in $O(n^2 v_{max})$ time, where v_{max} is the maximum of any value.

Pf. The optimal value $V^* \leq n v_{max}$.

There is one subproblem for each item and for each value $v \leq V^*$.

It takes $O(1)$ time per subproblem. □

Remark 1. Not polynomial in input size.

Remark 2. Polynomial time if values are small integers.

Knapsack Problem: PTAS

Intuition for approximation algorithm.

- Round all values up to lie in smaller range.
- Run dynamic programming algorithm II on rounded/scaled instance.
- Return the best of optimal items in rounded instance and the item with largest value.

item	value	weight
1	934221	1
2	5956342	2
3	17810013	5
4	21217800	6
5	27343199	7

original instance (W = 11)

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

rounded instance (W = 11)

Knapsack Problem: PTAS

Round up all values:

- v_{max} = largest value in original instance.
- ϵ = precision parameter.
- θ = scaling factor = $\epsilon v_{max}/n$.

$$\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta, \qquad \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$$

Obs. Optimal solutions to problem with $\bar{v}$ are equivalent to optimal solutions to problem with $\hat{v}$.

- $\bar{v}$ close to v so optimal solution using $\bar{v}$ is nearly optimal; $\hat{v}$ small and integral so dynamic programming algorithm II is fast.

Knapsack Problem: PTAS

Theorem 4
If S is solution found by the algorithm and S^* is any other feasible solution, then $(1+\epsilon)\sum_{i\in S} v_i \geq \sum_{i\in S^*} v_i$.

Pf. Let S' be the solution found by the rounding algorithm.

Let S^* be any feasible solution satisfying weight constraint.

$$\begin{aligned}
\sum_{i\in S^*} v_i &\leq \sum_{i\in S^*} \bar{v}_i \leq \sum_{i\in S'} \bar{v}_i \\
&\leq \sum_{i\in S'} (v_i+\theta) \leq \sum_{i\in S'} v_i + n\theta \\
&\leq \sum_{i\in S'} v_i + \epsilon v_{max} \\
&\leq (1+\epsilon)\sum_{i\in S} v_i. \; \square
\end{aligned}$$

Knapsack Problem: PTAS

Theorem 5

For any $\epsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1+\epsilon)$ factor of the optimum in $O(n^3/\epsilon)$ time.

Pf. We have already proved the accuracy bound.

Dynamic program II running time is $O(n^2 \hat{v}_{max})$, where

$$\hat{v}_{max} = \left\lceil \frac{v_{max}}{\theta} \right\rceil = \left\lceil \frac{n}{\epsilon} \right\rceil . \square$$

Load Balancing: LPT Rule Revisited

LPT Rule (Longest Processing Time)

- Order the job list by non-increasing processing times. That is, $t_1 \geq t_2 \geq \ldots \geq t_n$.
- Use List Scheduling (LS).

Theorem 6 (Graham 1969)

LPT rule is a $(\frac{4}{3} - \frac{1}{3m})$*-approximation algorithm, and the bound is tight.*

Load Balancing: LPT Rule Revisited

Pf. Denote the makespan due to LPT by $C_{\max}^{LPT}$, and job that completes last by J_k.

Case 1. $t_k \le C^*_{\max}/3$, from the analysis of LS

$$\begin{aligned} C_{\max}^{LPT} &\le \frac{1}{m}\sum_j t_j + \left(1 - \frac{1}{m}\right) t_k, \\ &\le \left(\frac{4}{3} - \frac{1}{3m}\right) C^*_{\max}. \end{aligned}$$

Case 2. $t_k > C^*_{\max}/3$, w.l.o.g. we can assume J_k was the last job scheduled.

In the optimal schedule at most two jobs can be processed on any machine.

It is not hard to see that the schedule returned by LPT rule is optimal. □

PTAS for Load Balancing: Analysis

PTAS for load balancing with constant machines

1. Schedule the $h \leq n$ longest jobs optimally; Denote the subset of jobs by B.
2. Apply LS to the remaining jobs.

- Assume the makespans produced in the two steps be $C^1_{\max}$ and $C^2_{\max}$.
- $C^1_{\max}$ is the makespan that results from the h longest jobs.
- $C^2_{\max}$ is the final makespan after scheduling the remaining jobs.

Load Balancing: PTAS for m Identical Machines

- We get an optimal solution if $C^2_{\max} \leq C^1_{\max}$.
- Assume that $C^2_{\max} > C^1_{\max}$, consider the last completed job J_k.
- We can see that (recall that $t_k \leq t_j, \ \forall j \in B$),

$$C^*_{\max} \geq \left(1 + \left\lfloor \frac{h}{m} \right\rfloor\right) t_k.$$

- By LS, we have

$$C_{\max} \leq C^*_{\max} + \left(1 - \frac{1}{m}\right) t_k.$$

- Combining the two results, we have

$$C_{\max} \leq C^*_{\max} + \left(\frac{1 - \frac{1}{m}}{1 + \left\lfloor \frac{h}{m} \right\rfloor}\right) C^*_{\max}.$$

Load Balancing: PTAS for *m* Identical Machines

Theorem 7

Given a fixed m, for the load balancing problem, a family of algorithms for different h's provides a PTAS, and the running time is $O\left(m^h\right)$.

Homework

- Read Chapter 11 of the textbook.
- Exercises 10 & 11 in Chapter 11.