Resource Allocation Algorithms

Lecture 07

Lecture by Dr. Tami Tamir
Typeset by Steven Karas

2018-05-08
Last edited 18:18:18 2018-05-08

Disclaimer These lecture notes are based on the lecture for the course Resource Allocation Algorithms, taught by Dr. Tami Tamir at IDC Herzliyah in the spring semester of 2017/2018. Sections may be based on the lecture slides written by Dr. Tami Tamir.

Agenda

- Nonapproximation of TSP
- Packing

1 Facility Location

1.1 Traveling Salesman Problem

Traveling salesman problem: shortest route that passes through all vertices in a complete graph. This is $\mathcal{NP}$-hard. A 2-approximation uses a double MST. A 1.5-approximation uses an MST and then cuts shortcuts between leaf nodes of the DFS.

1.1.1 Nonapproximation

We will prove that non-euclidean TSP is nonapproximable unless $P = \mathcal{NP}$. The proof follows by reduction from Hamilton Cycle. Assume we have an algorithm that provides a c-approximation for non-euclidean TSP. Given a graph $G = (V, E)$, construct $G' = (V, E')$ such that:

$$E' = \left\{ (u, v) = \begin{cases} c \cdot n & \text{if } (u, v) \notin E \\ 1 & \text{else} \end{cases} \right\}$$

If our algorithm provides a c-approximation, there exists a Hamilton cycle.

2 Packing

Packing problems have items (with sizes, weights, objective values, etc), bins (number, limited capacity, etc), and a set of constraints. Problems can be formulated as both decision and optimization problems.

2.1 Knapsack Problem

Classic application of dynamic programming. Pack a discrete set of items into a limited bin. The knapsack problem is $\mathcal{NP}$-hard, which can be shown by reduction from Partition. Knapsack is weakly $\mathcal{NP}$-hard, which means that it is solvable in polynomial time for unary inputs.

Greedy Algorithm By packing items according to their marginal utility, we can construct a solution to the problem within $O(n \log n)$. However, we can show this does not tightly approximate the optimal solution. For contradiction, assume that there exists some constant c such that this algorithm approximates knapsack by that ratio. Let $b_1 = 2, w_1 = 1$ and $b_2 = 2c, w_2 = 2c$ with a knapsack of size $2c$. The greedy algorithm always chooses to pack the first item, but the optimal packs the second.

Improved Greedy 2-approximation. As above, but attempt to include the most absolutely valuable item.

Full proof can be found on slide 9.

Exact Solution - Variant 1 Given an input of n items with knapsack size W, define a table M of size $(n+1) \times (W+1)$ where:

$$\begin{aligned} M_{0,x} &= 0 \\ M_{i,x<0} &= -\infty \\ M_{i,x} &= \max \begin{cases} M_{i-1,x} \\ M_{i-1,x-w_i} + b_i \end{cases} \end{aligned}$$

Each entry in this table represents the maximal utility by packing a subset of the first i items into a knapsack of size x.

Exact Solution - Variant 2 Given an input of n items with knapsack size W, define a table M of size $(n+1) \times (\sum_i b_i)$ where:

$$\begin{aligned} M_{0,0} &= 0 \\ M_{0,v} &= \infty \\ M_{i,v} &= \min \begin{cases} M_{i-1,v} \\ M_{i-1,v-b_i} + w_i \end{cases} \end{aligned}$$

Each entry in this table represents the minimum weight necessary to achieve a value of v using the first i items.

2.1.1 Fully Polynomial Time Approximation Scheme (FPTAS)

A PTAS is an algorithm which takes an additional parameter ε such that it provides a $1+\varepsilon$-approximation. The gist of the approach is to round up the utilities of each item and run variant 2 as above.

The dynamic programming solution for variant 2 has size as above. Note that $\sum_i b_i \leq n \cdot B$ where $B = \max b_i$. Therefore, we can say that variant 2 has running time $O(n^2 B)$.

For some $\varepsilon > 0$, denote the scaling factor as $k = \varepsilon \frac{B}{n}$. Round the utility values of each item up to the nearest multiple of the scaling factor:

$$b'_i = \left\lceil \frac{b_i}{k} \right\rceil k$$

The number of unique columns needed for the table is now:

$$n\frac{B}{k} = \frac{n^2}{\varepsilon}$$

There was a fully worked example done on the board that will be uploaded to the course site later. The proof is as follows: Let S^* be the set of items included in the optimal solution, and S be the set of items produced by the algorithm. Recall that our rounding means that items have grown by at most k.

2.2 Bin Packing

Bin packing is searching for a packing of items with size $\in (0,1)$ into bins of size 1. The objective is to minimize the number of bins. Note that an optimal solution must use at least $\lceil \sum_i a_i \rceil$.

2.2.1 Next-fit

2-approximation to bin packing. Open an active bin. For each item, place it in the active bin if it fits; otherwise open a new bin and place it there.

Assuming that Next-fit uses h bins, the sum of item sizes in adjacent bins is greater than 1. Therefore, $\sum_i a_i \geq h/2$. The approximation is tight: given an input $(1/2, 1/2n, 1/2, 1/2n, \ldots)$, the optimal solution uses $n+1$ bins, whereas Next-fit uses $2n$.

Other approximations are First-fit, where $h_{ff} \leq 1.7\text{OPT} + 2$, and First-fit-decreasing, where $h_{ffd} \leq 1.222\text{OPT} + 3$. The best approximation currently known is $(1+\delta)\text{OPT}$; there is no known additive error approximation $(\text{OPT} + c)$. However, constrained instances that arise often in practice have additive error algorithms.

2.2.2 Unit Fractions

A constrained instance of bin packing where all items are of the form $\frac{1}{i}$ for some integer i.

$$H(W) = \left\lceil \sum_{i \in W} \frac{1}{w_i} \right\rceil$$

Any-fit-decreasing provides a solution in $H(W) + 1$. Note that the optimal solution is at best $H(W)$. Sort the items in decreasing size, and allocate them to any bin that fits them, or a new bin, if none exists. The number of bins used is:

$$1 + \left\lceil \sum_{i} \frac{1}{w_i} \right\rceil \leq 1 + \text{OPT}$$

The proof of this follows from two things that hold after packing k items: there are at most $k-1$ non-full bins, and each of the bins is at least $1 - \frac{1}{k}$ full.

Denote the sorted sequence of items as:

$$W = \left(\left(\frac{1}{2}\right)^{n_2}, \ldots, \left(\frac{1}{c}\right)^{n_c} \right)$$

Where $c \geq 2$ and $n_i \geq 0$ for any $2 \leq i \leq c$. Assume Any-fit-decreasing uses h full bins and h' not-full bins. After packing all the items of size at least k', there are at most $k-1$ not-full bins.

The full proof will be uploaded to the site later.

References

[1] Michael L Pinedo. *Scheduling: theory, algorithms, and systems.* Springer, 2016.