6.854 Advanced Algorithms

Lecture 19: 10/20/2004 Lecturer: David Karger

Scribes: Sergiy Sidenko

Approximation Algorithms (continued)

19.1 Relative Approximation Algorithms

Since absolute approximation algorithms are known to exist for so few optimization problems, a better class of approximation algorithms to consider are relative approximation algorithms. Because they are so commonplace, we will refer to them simply as approximation algorithms.

Definition 1 ***An α-approximation algorithm*** *finds a solution of value at most $\alpha \cdot OPT(I)$ for a minimization problem and at least $OPT(I)/\alpha$ for a maximization problem ($\alpha \geq 1$).*

Note that although α can vary with the size of the input, we will only consider those cases in which it is a constant. To illustrate the design and analysis of α-approximation algorithm, let us consider the Parallel Machine Scheduling problem, a generic form of load balancing.

Parallel Machine Scheduling: Given m machines m_i and n jobs with processing times p_j, assign the jobs to the machines to minimize the load

$$\max_i \sum_{j \in i} p_j,$$

the time required for all machines to complete their assigned jobs. In scheduling notation, this problem is described as $\mathrm{P} \parallel \mathrm{C}_{\max}$.

A natural way to solve this problem is to use *greedy algorithm* called **list scheduling**.

Definition 2 *A* ***list scheduling*** *algorithm assigns jobs to machines by assigning each job to the least loaded machine.*

Note that the order in which the jobs are processed is not specified.

Analysis

To analyze the performance of list scheduling, we must somehow compare its solution for each instance I (call this solution $A(I)$) to the optimum $OPT(I)$. But we do not know how to obtain an analytical expression for $OPT(I)$. Nonetheless, if we can find a meaningful lower bound $LB(I)$ for $OPT(I)$ and can prove that $A(I) \leq \alpha \cdot LB(I)$ for some α, we then have

$$\begin{aligned} A(I) &\leq \alpha \cdot LB(I) \\ &\leq \alpha \cdot OPT(I). \end{aligned}$$

Using the idea of lower-bounding $OPT(I)$, we can now determine the performance of list scheduling.

Claim 1 *List scheduling is a $(2 - 1/m)$-approximation algorithm for Parallel Machine Scheduling.*

Proof:

Consider the following two lower bounds for the optimum load $OPT(I)$:

- the maximum processing time $p = \max_j p_j$,
- the average load $L = \sum_j p_j/m$.

The maximum processing time p is clearly lower bound, as the machine to which the corresponding job is assigned requires at least time p to complete its tasks. To see that the average load is a lower bound, note that if all of the machines could complete their assigned tasks in less than time L, the maximum load would be less than the average, which is a contradiction. Now suppose machine m_i has the maximum runtime $L = c_{\max}$, and let job j be the last job that was assigned to m_i. At the time job j was assigned, m_i must have had the minimum load (call it L_i), since list scheduling assigns each job to the least loaded machine. Thus,

$$\begin{aligned} \sum_{\text{machine i}} p_i &\geq mL_i + p_j \\ &\geq m(L - p_j) + p_j \end{aligned}$$

Therefore,

$$\begin{aligned} OPT(I) &\geq \tfrac{1}{m}\left(m(L - p_j) + p_j\right) \\ &= L - (1 - 1/m)p_j, \\ \text{then} & \\ L &\leq OPT(I) + (1 - 1/m)p_j \\ &\leq OPT(I) + (1 - 1/m)OPT(I) \\ &\leq (2 - 1/m)OPT(I). \end{aligned}$$

The solution returned by list scheduling is $c_{\max}$, and thus list scheduling is a $(2-1/m)$-approximation algorithm for Parallel Machine Scheduling.

The example with $m(m - 1)$ jobs of size 1 and one job of size m for m machines shows that we cannot do better than $(2 - 1/m)OPT(I)$.

■

19.2 Polynomial Approximation Schemes

The obvious question to now ask is how good an α we can obtain.

Definition 3 *A **polynomial approximation scheme (PAS)** is a set of algorithms $\{A_\varepsilon\}$ for which each A_ε is a polynomial-time $(1+\varepsilon)$-approximation algorithm.*

Thus, given any $\varepsilon > 0$, a PAS provides an algorithm that achieves a $(1+\varepsilon)$-approximation. In order to devise a PAS we can use the method called k-enumeration.

Definition 4 *An approximation algorithm using k**-enumeration** finds an optimal solution for the k most important elements in the problem and then uses an approximate polynomial-time method to solve the reminder of the problem.*

19.2.1 A Polynomial Approximation Scheme for Parallel Machine Scheduling

We can do the following:

- Enumerate all possible assignments of the k largest jobs.
- For each of these partial assignments, list schedule the remaining jobs.
- Return as the solution the assignment with the minimum load.

Note that in enumerating all possible assignments of the k largest jobs, the algorithm will always find the optimal assignment for these jobs. The following claim demonstrates that this algorithm provides us with a PAS.

Claim 2 *For any fixed m, k-enumeration yields a polynomial approximation scheme for Parallel Machine Scheduling.*

Proof:

Let us consider the machine m_i with maximum runtime $c_{\max}$ and the last job that m_i was assigned.

If this job is among the k largest, then it is scheduled optimally, and $c_{\max}$ equals $OPT(I)$.

If this job is not among the k largest, without loss of generality we may assume that it is the $(k+1)$th largest job with processing time p_{k+1}. Therefore,

$$A(I) \leq OPT(I) + p_{k+1}.$$

However, $OPT(I)$ can be bound from below in the following way:

$$OPT(I) \geq \frac{kp_k}{m},$$

because $\frac{kp_k}{m}$ is the minimum average load when the largest k jobs have been scheduled.

Now we have:

$$\begin{aligned} A(I) &\leq OPT(I) + p_{k+1} \\ &\leq OPT(I) + OPT(I)\tfrac{m}{k} \\ &= OPT(I)\left(1 + \tfrac{m}{k}\right). \end{aligned}$$

Given $\varepsilon > 0$, if we let k equal m/ε, we will get

$$c_{\max} \leq (1+\varepsilon)OPT(I).$$

Finally, to determine the running time of the algorithm, note that because each of the k largest jobs can be assigned to any of the m machines, there are $m^k = m^{m/\varepsilon}$ possible assignments of these jobs. Since the list scheduling performed for each of these assignments takes $O(n)$ time, the total running time is $O(nm^{m/\epsilon})$, which is polynomial because m is fixed. Thus, given an $\varepsilon > 0$, the algorithm is a $(1+\varepsilon)$-approximation, and so we have a polynomial approximation scheme.

■

19.3 Fully Polynomial Approximation Schemes

Consider the PAS in the previous section for P $\|$ $C_{\max}$. The running time for the algorithm is prohibitive even for moderate values of ε. The next level of improvement, therefore, would be approximation algorithms that run in time polynomial in $1/\varepsilon$, leading to the definition below.

Definition 5 ***Fully Polynomial Approximation Scheme (FPAS)*** *is a set of approximation algorithms such that each algorithm $A(\varepsilon)$ in this set runs in time that is polynomial in the size of the input, as well as in $1/\varepsilon$.*

There are few NP-complete problems that allow for FPAS. Below we discuss FPAS for the Knapsack problem.

19.3.1 A Fully Polynomial Approximation Scheme for the Knapsack Problem

The Knapsack problem receives as input an instance I of n items with profits p_i, sizes s_i and knapsack size (or capacity) B. The output of the Knapsack problem is the subset S of items of total size at most B, and that has profit:

$$\max \sum_{i \in S} p_i.$$

Suppose now that the profits are integers; then we can write a DP algorithm based on the minimum size subset with profit p for items 1, 2, ..., r as follows:

$$M(r, p) = \min \{M(r-1, p), M(r-1, p-p_r) + s_r\}.$$

The corresponding table of values can be filled in $O\left(n \sum_i p_i\right)$ (note that this is not FPAS in itself).

Now, we consider the general case where the profits are not assumed to be integers. Once again, we use a rounding technique but one that can be considered a generic approach for developing FPAS for other NP-complete problems that allows for FPAS. Suppose we multiplied all profits p_i by $n/\varepsilon \cdot OPT$; then the new optimal objective value is apparently n/ε. Now, we can round the profits down to the nearest integer, and hence the optimal objective value decreases at most by n; expressed differently, the decrease in objective value is at most $\varepsilon \cdot OPT$. Using the DP algorithm above, we can therefore find the optimal solution to the rounded problem in $O(n^2/\varepsilon)$, thus providing us with FPAS in $1/\varepsilon$.