6.854 Advanced Algorithms

Lecture 13: 10/22/2003 Lecturer: Erik Demaine, David Karger

Scribes: S. Gurumurthi

Approximation Algorithms (Continued...)

13.1 Introduction and Review

In the previous lecture we examined the notion of NP-completeness, and the techniques to handle NP-completeness. Specific examples of NP-complete problems were mentioned including:

1. Satisfiability Problem (**SAT**)
2. Bin Packing
3. Maximum Independent Set (**MIS**)
4. Knapsack Problem
5. Parallel Machine Scheduling ($P \| C_{max}$)
6. Traveling Salesman Problem (**TSP**)

We noted that these problems cannot be solved efficiently given the assumption that $P \neq NP$. As a work-around to the NP-complete decision problems and the corresponding NP-hard optimization problems, approximation algorithms attempt to derive **approximate** and **polynomial-time** solutions for any specific problem. After defining terminology for describing generic optimization problems in this context, we defined the various types of approximation schemes including:

1. **Absolute approximation** schemes that find a solution to a minimization problem, of value at most $OPT(I) + \alpha$.
2. **Relative approximation** schemes that find a solution to a minimization problem, of value at most $\alpha.OPT(I)$.
3. **Polynomial approximation** schemes (**PAS**), that for any given positive ϵ, find an algorithm that achieves a (1+ϵ)-approximation.

Illustrative examples of the absolute approximation scheme were discussed in the form of the **Graph Coloring** and **Edge Coloring** problems. Specifically, it was shown using simple structural arguments that a 2-*absolute approximation* scheme exists for the planar graph coloring problem (where we have to color the nodes in the graph, so that adjecent nodes never have the same color). Similarly, for the edge coloring problem, a 1-*absolute approximation* scheme was shown to exist. It was noted, on the same token, that most NP-hard optimization problems do not allow for an absolute approximation scheme; examples of negative results were shown using **scaling** techniques.

Graham's **list scheduling** algorithm for the $P \| C_{max}$ problem was discussed as an example of relative approximation schemes. In the $P \| C_{max}$ problem, given m machines m_i, and n jobs with processing times p_j, we have to assign the jobs to the machines, so as to minimize the objective function:

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

List scheduling assigns, consecutively, each job to the least loaded machine until all the jobs are assigned; this was shown to be a 2-*approximation* scheme for $P||C_{max}$ for the case where we assign jobs in any random order.

Finally to illustrate **PAS**, a $k - enumeration$ approximation algorithm for $P||C_{max}$was discussed, where for each possible assignment of the k largest jobs to the m machines, we invoke list scheduling to arrive at a solution with the lowest objective value. For any fixed m, $k - enumeration$ was shown to be a polynomial approximation scheme. It was noted that the running time of the $k - enumeration$ scheme was $O(nm^k)$=$O(nm^{m/\epsilon})$, and therefore depends on m. However, for certain classes of the input I for the $P||C_{max}$ problem, specifically where there are only k (bounded) distinct sizes for the jobs, a dynamic programming formulation was discussed, that computes an *optimal* schedule in polynomial time. This algorithm is discussed in the next section.

In Section 13.3, we introduce the fully polynomial approximation scheme (FPAS) with an example of the Knapsack problem.

13.2 A Polynomial Time Approximation Scheme (PAS) for $P||C_{max}$

Our goal in this section is to find a polynomial time $(1+\epsilon)$-approximation scheme for the $P||C_{max}$ problem that runs in polynomial time. Towards this end, we examine first a special case of the problem; consider $P||C_{max}$ where there are only k (bounded) distinct sizes for the jobs. Every instance of the input can be characterized by the number of jobs of each type, called its **profile**. The maximm possible number of profiles is n^k, and is therefore polynomial in the number of jobs. Suppose our first question is: what is the minimum number of machines needed to complete the given profile in time T. To answer this question, we construct a dynamic program (DP) as follows:

Step 1: Enumerate the set Q of all the profiles that can be completed by a single machine in time T. Thus, for any profile $(x_1, x_2, ..., x_k) \in Q$, we know that the minimum number of machines needed to complete this profile, call it $M(x_1, x_2, ..., x_k)$, equals 1.
Step 2: We are interested in computing $M(a_1, a_2, ..., a_k)$, where $(a_1, a_2, ..., a_k)$ is the input profile of jobs. For this, we define the recursion:

$$M(a_1, a_2, ..., a_k) = 1 + \min_{(x_1, x_2, ..., x_k) \in Q} M(a_1 - x_1, a_2 - x_2, ..., a_k - x_k).$$

with the running time of the DP algorithm of order $O(n^{2k})$.
Step 3: From *Step* 2 above, we have $M(a_1, a_2, ..., a_k)$ for any T. Then using the DP procedure in a binary search over the values of T, it can be shown that the optimal schedule can be determined in polynomial time.

Next we make the following assumptions about the general case of $P||C_{max}$with unrestricted job types:

1. ϵT, $\epsilon^2 T$, $1/\epsilon$, and $1/\epsilon^2$ are integers.
2. Consider jobs of size greater than ϵT as "large" jobs, with the view that smaller jobs will not contribute much to the make-span.

If there are some "small" jobs in the input, based on the above assumptions, then we use the DP algorithm above to determine if the profile of large jobs can be completed in time T. In this case, we attempt to use list scheduling to add the remaining small jobs to this optimal large-job schedule while maintaining the $(1+\epsilon)T$ upper limit on the load on any single machine. If we fail to find such a schedule that completes in time at most $(1+\epsilon)T$, then we try the next value of T in the binary search.

On the other hand, if all the jobs in the input qualify as large jobs, then we scale the p_i's down to the nearest integer multiple of $\epsilon^2 T$. Now, we use the DP procedure to determine an optimal schedule for the re-scaled input that completes within time T. The claim here is that a schedule that is optimal for the re-scaled input is also optimal for the original input. To see this, note that the p_i's can be scaled down at most by $\epsilon^2 T$, and that there can only be an integer number $1/\epsilon^2$ of re-scaled job types. Further observing that by adopting this optimal schedule, the load on any given machine increases (reverting to the original input) by at most ϵT, we see that we have a **PAS** for $P||C_{max}$.

13.3 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 $1/\epsilon$, leading to the definition below.

Definition 1 ***Fully Polynomial Approximation Schemes (FPAS):*** **FPAS** *are a set of approximation schemes $A(\epsilon)$ such that each algorithm in this set runs in time that is polynomial in the size of the input,* as well as in $1/\epsilon$.

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

13.3.1 FPAS 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:

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

The corresponding table of values can be filled in $O(n. \sum_i p_i)$ (note however 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 allow for **FPAS**. Suppose we multiplied all profits p_i by a $n/\epsilon.OPT$; then the new optimal objective value is 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 $\epsilon.Optimum$. Using the DP algorithm above, we can therefore find the optimal solution to the rounded problem in $O(n^2/\epsilon)$, thus providing us with **FPAS** in $1/\epsilon$.

13.3.2 Note on FPAS and Weakly NP-Hard Problems

We say that a problem is *weakly* NP-hard if it is NP-hard but P when the numbers involved are written in unary code. The following lemma is therefore of interest in the context of approximation algorithms.

Lemma 1 *If a problem allows for* **FPAS** *then it is weakly NP-hard; on the other hand if a problem is weakly NP-hard then this approximately implies the existence of* **FPAS**.

13.4 Negative Results for PAS

Many problems do not allow for **PAS**, and so we would like to be able to show this definitively. We use so-called "gap" arguments to show this for many problems: since it is impossible to distinguish between OPT=a, and OPT=b, we must have no better than a b/a-approximation.

For example: we can claim that there does not exist a 3/2-approximation (i.e. a **PAS** for bin-packing (the proof lies in the fact that the **Partition** problem is NP-complete). That said, there is another class of approximation schemes called the **Asymptotic PAS / FPAS** (**APAS / AFPAS**) that have the property that the objective value of the approximation is at most $(1+\epsilon).OPT + O(OPT)$. For bin-packing, APAS was shown by Vega and Lueker in 1981, whereas Karmarkar and Karp have shown AFPAS for bin-packing in 1982.

An open problem in this area is to show a polynomial time asymptotic approximation scheme that runs in $(1+\epsilon).OPT + O(1)$, since there is no reason to believe otherwise.

13.5 Approximation Algorithms For MAX-CUT, Set Cover and Vertex Cover Problems

$MAX - SNP$ problems are approximable to some constant factor, whereas $MAX - SNP$-complete problems are not approximable to some other constant factor. For example, consider the **MAX-CUT** problem, where we wish to partition the vertices of a graph into two sets, in order to maximize the crossing edges. A greedy algorithm is where we consider the vertices in order, and place each vertex in the set with the fewer placed neighbors. The claim is that this is a 2-approximation scheme, i.e. the approximate cut containts at least half the edges of the maximum cut. The proof lies in the fact that each time a vertex is placed, at least 1/2 the edges that get determined are cut. This 2-approximation was state-of-the-art for decades until Michel Goemans recently developed a (1/0.878)-approximation.

In **Set Cover** problems, we have a collection of sets over some ground set, i.e. $S_1, S_2, ..., S_n$ $\subseteq$S;$\|S\| = m$. We seek the minimum collection of sets covering the entire ground set. The grredy algorithm for this problem at each step, picks the set that containts the largest number of elements not already covered. The greedy algorithm can be shown to be an $O(\log m)$-approximation.

In **Vertex Cover** problems, we simply pick the vertices to cover all the edges. The obvious greedy algorithm is to pick a vertex with the most number of nieghbors– however this provides for a bad approximation. Another greedy algorithm is to take any edge and discarding incident edges, to pick both its end-points (vertices); we repeat the process until we have exhausted all the edges. The rationale is that know OPT has to have at least one of those two vertices in it. To analyze, note that each time we take an edge, we essentially take one vertex from OPT; since the number of edges taken is at most OPT, the number of vertices I take is at most $2.OPT$. It is known that we cannot get better than a (7/6)-approximation for **Vertex Cover** problems.