College of Computer & Information Science
Northeastern University
CS G399: Algorithmic Power Tools I

Fall 2007
7 September 2007
Scribe: Rajmohan Rajaraman

Lecture Outline:

- Course outline
- Brief background on NP-completeness
- Approximation algorithms
- Set cover
- Greedy algorithm for set cover

In this lecture, we have a quick review of NP-completeness, and then introduce approximation algorithms using the set cover problem. Good references for the material covered in this lecture are the classic text on algorithms [CLRS01], a nice set of lecture notes prepared by Rajeev Motwani [Mot92], and an excellent text on approximation algorithms by Vijay Vazirani [Vaz03].

1 Approximation algorithms

A major focus of this course is on the study of hard optimization problems. Many of these problems fall in the class of problems referred to as NP-complete. It is widely believed that no NP-complete problem can be solved optimally in polynomial time. We can relax our requirements in several ways:

- **Super-polynomial time algorithms:** Instead of requiring poly-time algorithms, we may demand a slightly super-polynomial time algorithm. Indeed, for certain problems such as Knapsack, we can obtain efficient slightly super-polynomial time algorithms that solve the probem optimally. It turns out, however, that the fastest known algorithms for almost all of the interesting NP-complete problems is exponential time.
- **Focus on a subset of the instances:** We can also relax our objective by focusing on a subset of the instances, rather than demanding that we solve every instance optimally. The trouble with this approach is to define the particular subset of instances we are interested in. While it may be possible to define the notion of a "random instance" in some cases and analyze the expected running-time of algorithms by probabilistic means, this is difficult to justify for most problems.
- **Fixed-parameter tractability:** Another approach that has gained attention recently for certain problems is to study the fixed-parameter complexity of the problem. For some problems, it may be possible to devise algorithms that are polynomial in the input size and superpolynomial in a parameter of the problem that is usually small for many applications. For example, there exists an algorithm for the vertex cover problem in time $O(kn + 1.274^k)$, where n is the number of nodes in the vertex cover graph and k is the size of the vertex cover found. If the size of the min-vertex cover is small, then the preceding algorithm could be efficient. Even though the area is fairly new (since 1999), there is already a book out [Nie06].

- **Approximation algorithms:** A fourth relaxation, which is the one we will discuss during part of this course, is to allow for approximate solutions. That is, rather than solving optimally for every instance, we will demand *polynomial-time* algorithms that solve *near-optimally* for *every instance*.

What does "near-optimal" mean? One notion of approximation is that of an *absolute performance guarantee*, in which the value of the solution returned by the approximation algorithm differs from the optimal value by an absolute constant. Let Π be an optimization problem and let I be an instance of Π. Given an algorithm $\mathcal{A}$ for Π, let $\mathcal{A}(I)$ denote both the solution as well as the value of the solution returned by $\mathcal{A}$ on instance I. Also, let $\mathrm{OPT}(I)$ denote the optimal value for instance I.

Definition 1. An approximation algorithm $\mathcal{A}$ for problem Π has an **absolute performance guarantee** of k if for the following condition holds for all instances I of Π:

$$|\mathcal{A}(I) - \mathrm{OPT}(I)| \leq k.$$

□

While this notion is useful for certain problems such as the minimum-degree spanning tree problem, it turns out to be inappropriate for most NP-complete problems. The notion of approximation that is most widely used is that of a *relative performance guarantee*, which we now define.

Definition 2. An approximation algorithm $\mathcal{A}$ for problem Π has an **approximation ratio** of r if the following condition holds for all instances I of Π:

$$\frac{\mathcal{A}(I)}{\mathrm{OPT}I} = r.$$

2 Set cover

The set cover problem plays a central role in the study of approximation algorithms. It has numerous applications in diverse areas and several other fundamental optimization problems are natural generalizations of set cover. It is also a very useful problem to demonstrate several fundamental techniques developed in the area of approximation algorithms.

Problem 1. Given a universe $\mathcal{U} = \{e_1, \ldots, e_n\}$ of n elements, a collection $\mathcal{S}$ of m subsets of $\mathcal{U}$, and a cost function $c : \mathcal{S} \to Q^+$, the set cover problem seeks a minimum-cost subset of $\mathcal{S}$ that covers all elements of $\mathcal{U}$.

The set cover problem is NP-complete, so we seek good approximation algorithms. The first algorithm that comes to mind for the set cover problem is the following greedy algorithm.

Repeat the following step until all elements of $\mathcal{U}$ are covered: select a set $S \in \mathcal{S}$ that has the least ratio of cost to the number of uncovered elements in S.

It is relatively easy to see that the greedy algorithm is not optimal. How far from optimal can it be? Here is a preliminary analysis. Let OPT denote the optimal solution to a given instance and

let $c(\text{OPT})$ denote the cost of OPT. At any instant, define the normalized cost of a set S to be the ratio of $c(S)$ to the number of uncovered elements in S.

First Analysis: Divide the algorithm's progress into phases, starting from phase 0. In the ith phase, the number of uncovered elements is in $[n/2^i, n/2^{i+1})$. Therefore, the normalized cost of sets selected in the ith phase is always at most $c(\text{OPT})/(n/2^{i+1})$. Since the number of elements covered in the ith phase is at most $n/2^i$, the cost of the sets added in the ith phase is at most $2c(\text{OPT})$. The total number of phases is at most $\lg n$. Therefore, the total cost of the greedy solution is at most $2 \lg n c(\text{OPT})$, establishing that the approximation ratio of the greedy algorithm is at most $2 \lg n$.

Second Analysis: A more accurate analysis can be given by the following direct argument. Let S_1 denote the first set selected by the greedy algorithm, and let OPT denote the cost of the optimal solution. We claim that $c(S_1)/|S_1|$ is at most $c(\text{OPT})/n$. This is because if every set S of the optimum solution had $c(S)/|S|$ exceed $c(\text{OPT})/n$, then the total cost of the optimum solution, which equals $\sum_{s \in \text{OPT}} c(S)$ exceeds $c(\text{OPT}) \sum_{S \in \text{OPT}} |S| > \text{OPT}$, a contradiction.

We can use the same argument to derive that if S_2 is the second set selected by the greedy algorithm, then $c(S_2)/|S_2 - S_1|$ is at most $c(\text{OPT})/(n - |S_1|)$. In general, if S_i is the ith set selected, we obtain

$$\frac{c(S_i)}{|S_i - \cup_{j<i} S_j|} \leq \frac{\text{OPT}}{n - \cup_{j<i} S_j}.$$

Summing over all i, we obtain that the cost of the greedy algorithm is at most

$$\begin{aligned} & c(\text{OPT}) \left(\frac{|S_1|}{n} + \frac{|S_2 - S_1|}{n - |S_1|} + \frac{|S_3 - (S_1 \cup S_2)|}{n - |S_1 \cup S_2|} + \cdots + \frac{|S_i - \cup_{j<i} S_j|}{n - |\cup_{j<i} S_j|} + \cdots \right) \\ \leq \; & c(\text{OPT}) \left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{1} \right) \\ = \; & c(\text{OPT}) H_n. \end{aligned}$$

Third Analysis: Here is another slicker proof that the greedy algorithm has approximation ratio H_n.

Define the price of an element to be the average cost at which it is covered. Let $e_1, e_2, \ldots, e_n$ denote the elements in the order they were covered.

Lemma 1. *The price of element e_i is at most $OPT/(n - i + 1)$.*

Proof. At the instant element e_i is covered, there are at least $n - i + 1$ uncovered elements. Therefore, there exists a set in the optimal solution that has a cost-effectiveness of at most $OPT/(n - i + 1)$. Since e_i is covered in this step, its price is at most $OPT/(n - i + 1)$. □

Theorem 1. *The approximation ratio of the greedy algorithm is at most H_n.*

Proof. The cost of a selected set is simply the sum of the prices of the new elements it covers. Thus, the cost of the greedy solution is simply the sum of the prices of all the elements, which, according to Lemma 1, is upper bounded by

$$OPT \sum_{i=1}^{n} \frac{1}{n - i + 1} = OPT \cdot H_n.$$

□

References

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 2001.

[Mot92] R. Motwani. Lecture notes on approximation algorithms: Volume I. Technical Report STAN-CS-92-1435, Department of Computer Science, Stanford University, 1992. Postscript document is available at http://Theory.Stanford.EDU/~rajeev/postscripts/approximations.ps.gz (134 pages).

[Nie06] R. Niedermeier, editor. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

[Vaz03] V. Vazirani. *Approximation Algorithms.* Springer-Verlag, 2003.