CMSC 858K Advanced Algorithms	Lecture 01
Lecturer: Samir Khuller	Jan 28, 1999

Original Notes by Charles Lin

10 Approximation Algorithms: Set Cover

10.1 Introduction

The goal of algorithm design is to develop efficient algorithms to solve problems. An algorithm is usually considered efficient if its worst case running time on an instance of a problem is polynomial in the size of the problem. However, there are problems where the best algorithms have running times which are exponential (or worse) in the worst case. Most notably, the class of NP-hard problems currently have algorithms which are exponential in the worst case.

The need to develop efficient algorithms has lead to the study of approximation algorithms. By relaxing the optimality of the solution, polynomial algorithms have been developed that solve NP complete problems to within a constant factor of an optimal solution.

10.2 Approximation Algorithms

We can formalize the notion of an approximation algorithm some more by considering what it means to approximate a minimization problem. A minimization problem is one where the solution minimizes some quantity under certain criteria. For example, vertex cover is usually considered a minimization problem. Given a graph G, the goal is to find the fewest vertices that "cover" all edges.

Let P be the set of problem instances for a minimization problem. Let OPT be the optimal algorithm for solving the problems in P. If the problem is vertex cover, then P contains all undirected graphs, and OPT would find the minimum vertex cover for each of the instances. Let $I \in P$ be a problem instance. Then, we will say that $OPT(I)$ is the minimum solution for I. For vertex cover, we can say that this is the size of the minimum vertex cover.

An α-approximation algorithm A (we often refer to α as the approximation factor), approximates the optimal solution as follows:

$$\forall I,\ A(I) \leq \alpha\ OPT(I) \qquad \text{where}\ \alpha > 1.$$

If there were a 2-approximation algorithm for vertex cover and the optimal solution for a given graph had 3 vertices, then the approximate solution would do no worse than find a vertex cover with 6 vertices. Note that an α-approximation algorithm only gives an upper bound to how far away the approximation is from optimal. While there may exist problem instances as bad as α times the optimal, the algorithm may produce solutions close to optimal for many problem instances.

Some α-approximation algorithms have α depend on the size of the input instance or some other property of the input, whereas others produce approximations independent of the input size. Later on, for example, we will consider an $H(|S_{max}|)$-approximation algorithm where the approximation factor is a function of the maximum set size in a collection of sets.

For a maximization problem, an α-approximation algorithm would satisfy the following formula:

$$\forall I,\ OPT(I) \geq A(I) \geq \alpha\ OPT(I) \qquad \text{where}\ \alpha < 1.$$

10.3 Polynomial Approximation Schemes

For some problems, there are polynomial approximation schemes (PAS). PAS are actually a family of algorithms that approximate the solution to a given problem *as well as we wish.* However, the closer the solution is to the optimal, the longer it takes to find it. More formally, for any $\epsilon > 0$, there is an approximation algorithm, A_ϵ such that

$$\forall I,\ A_\epsilon(I) \leq (1+\epsilon)\ OPT(I)$$

There is a similar formula for maximization problems.

Although a problem may have a PAS, these schemes can have a very bad dependence on ϵ. A PAS can produce algorithms that are exponential in $\frac{1}{\epsilon}$, e.g., $O(n^{\frac{1}{\epsilon}})$. Notice that for any fixed ϵ, the complexity is polynomial. However, as ϵ approaches zero, the polynomial grows very quickly. A fully polynomial approximation scheme (FPAS) is one whose complexity is polynomial in both n (the problem size) and $\frac{1}{\epsilon}$. Very few problems have a FPAS and those that do are often impractical because of their dependence on ϵ. Bin packing is one such example.

10.4 Set Cover

We will start by considering approximation algorithms for set cover, which is a very fundamental NP-complete problem, as well as problems related to set cover.

The input to set cover are two sets, X and S.

$$\begin{aligned} X &= \{x_1, x_2, \ldots, x_n\} \\ S &= \{S_1, S_2, \ldots, S_m\} \qquad \text{where } S_i \subseteq X \end{aligned}$$

We further assume that $\bigcup_{S_i \in S} S_i = X$. Hence, each element of X is in some set S_i. A *set cover* is a subset $S' \subseteq S$ such that

$$\bigcup_{S_j \in S'} S_j = X$$

The set cover problem attempts to find the minimum number of sets that cover the elements in X. A variation of the problem assigns weights to each set of S. An optimal solution for this problem finds a set cover with the least total weight. The standard set cover problem can then be seen as a weighted set cover problem where the weights of each set are 1.

The set cover problem can also be described as the *Hitting Set Problem.* The formulation of the hitting set problem is different from set cover, but the problems are equivalent. In the hitting set problem, we have a hypergraph, $H = (V, E)$ where

$$\begin{aligned} V &= \{v_1, v_2, \ldots, v_n\} \\ E &= \{e_1, e_2, \ldots, e_m\} \end{aligned}$$

In a normal graph, an edge is incident on two vertices. In a hypergraph, an edge can be incident on one or more vertices. We can represent edges in a hypergraph as subset of vertices. Hence, $e_i \subseteq V$ for all $e_i \in E$. In the Hitting Set Problem, the goal is to pick the fewest vertices which "cover" all hyperedges in H.

To show that that hitting set is equivalent to set cover, we can transform an instance of hitting set to that of set cover.

$$X = E$$

$$S = \{S_1, S_2, \ldots, S_n\}$$

where $e \in S_i$ iff edge $e \in E$ touches vertex v_i. For every vertex v_i in V, there is a corresponding set S_i in S. The reverse transformation from set cover to hitting set is similar.

The Hitting Set Problem can be seen as a generalization of the vertex cover problem on graphs, to hypergraphs. The two problems are exactly the same except that vertex cover adds the constraint that edges touch exactly two vertices. Vertex cover can also be seen as a specialization of set cover where each element of X appears in exactly two sets in S.

10.5 Approximating Set Cover

We will consider an approximation algorithm for set cover whose value for α is $H(|S_{max}|)$ where S_{max} is the largest set in S. H is the harmonic function and is defined as

$$H(d) = \sum_{i=1}^{d} \frac{1}{i}.$$

A simple heuristic is to greedily pick sets from S. At each step, pick $S_j \in S$ which contains the most number of uncovered vertices. Repeat until all vertices are covered. The following is pseudo-code for that greedy algorithm.

```
Greedy-Set-Cover(X, S)
U ← X      // U is the set of uncovered elements
S' ← ∅     // S' is the current set cover
while U ≠ ∅ do
    pick S_j ∈ S such that |S_j ∩ U| is maximized    // pick set with most uncovered elements
    S' ← S' ∪ S_j
    U ← U − S_j
end
```

To show that the approximation factor is $H(|S_{max}|)$, we will use a "charging" scheme. The cost of choosing a set is one (dollar). If we pick a set with k uncovered elements, each of the uncovered elements will be charged $\frac{1}{k}$ dollars. Since each element is covered only once, it is charged only once. The cost of the greedy algorithm's solution will be the total charge on all the elements.

$$\text{Greedy Cost} = \sum_{x_i \in X} c(x_i) \leq \sum_{S_i \in OPT} C(S_i) \leq |OPT| \cdot H(|S_{max}|)$$

where $c(x_i)$ is the cost assigned to element x_i. S_i is a set in the optimum set cover. $C(S_i)$ is the cost of set S_i by summing the charges of the elements in S_i. That is,

$$C(S_i) = \sum_{x_i \in S_i} c(x_i)$$

$c(x_i)$ is still the cost assigned to x_i from the greedy algorithm.

The inequality, $\sum_{x_i \in X} c(x_i) \leq \sum_{S_i \in OPT} C(S_i)$, says that the cost of the optimum set cover is at least the sum of the cost of the elements. We can see this more readily if we rewrite the cost of the optimum set cover as

$$\sum_{S_i \in OPT} C(S_i) = \sum_{x_i \in X} c(x_i) \cdot n(x_i)$$

where $n(x_i)$ is the number of times element x_i appears in the optimum set cover. Since each element appears at least once, the inequality naturally follows.

To show the second inequality, $\sum_{S_i \in OPT} C(S_i) \leq |OPT| \cdot H(|S_{max}|)$, it suffices to show that $C(S_i) \leq H(|S_i|)$. The inequality would then hold as follows:

$$\sum_{S_i \in OPT} C(S_i) \leq \sum_{S_i \in OPT} H(|S_i|) \leq \sum_{S_i \in OPT} H(|S_{max}|) = |OPT| \cdot H(|S_{max}|).$$

To show $C(S_i) \leq H(|S_i|)$, consider an arbitrary set $S_i \in S$. As sets are being picked by the greedy algorithm, we want to see what happens to the elements of S_i. Let U_0 be the set of uncovered elements in S_i when the algorithm starts. As sets are picked, elements of S_i will be covered a piece at a time. Suppose we are in the middle of running the algorithm. After j iterations, U_j represents the uncovered elements of S_i. If the algorithm picks a set which covers elements in U_j, we write the new set of uncovered elements as U_{j+1}. Note that U_{j+1} is a proper subset of U_j. Hence, $U_0, U_1, \ldots, U_k, U_{k+1}$ is the history of uncovered elements of S_i, where $U_{k+1} = \emptyset$.

The following illustrates the history of uncovered elements in S_i.

$$U_0 \xrightarrow{|U_0 - U_1|} U_1 \xrightarrow{|U_1 - U_2|} U_2 \ldots U_k \xrightarrow{|U_k|} U_{k+1} = \emptyset$$

Above each arrow is the number of elements that were charged. Going from U_j to U_{j+1}, the number of elements charged is $|U_j - U_{j+1}|$. How much are these elements charged?

Suppose we are at the point in the algorithm where U_j represents the set of uncovered elements in S_i, and the greedy algorithm is just about to pick a set. Assume that it picks a set which covers at least one uncovered element in S_i (otherwise, let the algorithm continue until this happens).

We want to show that the elements that get covered are charged no more than $1/|U_j|$. If it picks S_i, then each element in U_j is charged exactly $1/|U_j|$. Let's assume it doesn't pick S_i, but picks T instead. The only reason it would pick T is if T had at least $|U_j|$ uncovered elements. In this case, each of uncovered elements in T is charged at most $1/|T| \leq 1/|U_j|$.

By assumption, we said some elements of T overlap uncovered elements in S_i. Specifically, $T \cap U_j$ was covered when T was picked. The total additional cost to S_i from picking T is at most

$$|T \cap U_j| \cdot \frac{1}{|U_j|} = (U_j - U_{j+1}) \cdot \frac{1}{|U_j|}$$

The total cost charged to S_i is therefore

$$C(S_i) \leq \frac{|U_0 - U_1|}{|U_0|} + \frac{|U_1 - U_2|}{|U_1|} + \ldots + \frac{|U_k - U_{k+1}|}{|U_k|}$$

Since $U_{k+1} = \emptyset$, the last term simplifies to 1.

Each of the terms of sum have the form $(x-y)/x$ for $x > y$. The claim is that this is less than $H(x)-H(y)$. This can be shown by expanding $H(x)$ and $H(y)$.

$$\begin{array}{rcl} \text{Consider} & & H(x) - H(y) \\ & = & (1 + \frac{1}{2} + \ldots + \frac{1}{x}) - (1 + \frac{1}{2} + \ldots + \frac{1}{y}) \\ & = & \frac{1}{y+1} + \frac{1}{y+2} + \ldots \frac{1}{x} \\ & \geq & (x-y) \cdot \frac{1}{x} \end{array}$$

The last equation comes from the fact that line 3 contains $x - y$ terms, each term being at least $1/x$.

Using this fact, we can rewrite the total charge on S_i as

$$\begin{array}{rcl} C(S_i) & \leq & (H(|U_0|) - H(|U_1|)) + (H(|U_1|) - H(|U_2|)) + \ldots + (H(|U_k|) - H(|U_{k+1}|)) \\ & = & H(|U_0|) - H(|U_{k+1}|) \\ & = & H(|U_0|) \end{array}$$

Since U_0 is just S_i, then $C(S_i) \leq H(|S_i|)$.

One might ask how tight can $C(S_i)$ be made? Suppose S_i contains 5 elements. In one step 3 elements are covered and in a subsequent step 2 more elements are covered. The cost of the set is at most $\frac{1}{5} + \frac{1}{5} + \frac{1}{5} + \frac{1}{2} + \frac{1}{2}$ which is less than $H(5)$. Can we make $C(S_i)$ exactly equal to $H(|S_i|)$? Yes, but only if at most one element is covered in S_i whenever the greedy algorithm picks a set to be added to the set cover.

Here is a scenario where S_0 can be charged exactly $H(|S_0|)$. Let $S = \{S_0, S_1, \ldots, S_m\}$. Let $|S_j| = j$ (the size of the set is the subscript) except for S_0 which has size m. Let $S_0 = \{x_1, \ldots, x_m\}$. The sets $S_1, \ldots, S_m$ will have no elements in common. Each set only has one element in common with S_0. Specifically, $S_j \cap S_0 = \{x_j\}$ where $j > 0$.

If the greedy algorithm picks its set in the following order, $S_m, S_{m-1}, \ldots, S_1$, then $C(S_0)$ will be exactly $H(|S_0|)$.