CS880: Approximations Algorithms

Scribe: Matt Elder **Lecturer:** Shuchi Chawla

Topic: Greedy Approximations: Set Cover and Min Makespan **Date:** 1/30/06

3.1 Set Cover

The *Set Cover* problem is: Given a set of elements $E = \{e_1, e_2, \ldots, e_n\}$ and a set of m subsets of $E, \mathcal{S} = \{S_1, S_2, \ldots, S_n\}$, find a "least cost" collection C of sets from $\mathcal{S}$ such that C covers all elements in E. That is, $\cup_{S_i \in C} S_i = E$.

Set Cover comes in two flavors, unweighted and weighted. In unweighted Set Cover, the cost of a collection C is number of sets contained in it. In weighted Set Cover, there is a nonnegative weight function $w : \mathcal{S} \to \mathbb{R}$, and the cost of C is defined to be its total weight, i.e., $\sum_{S_i \in C} w(S_i)$.

First, we will deal with the unweighted Set Cover problem. The following algorithm is an extension of the greedy vertex cover algorithm that we discussed in Lecture 1.

Algorithm 3.1.1 Set Cover(E, $\mathcal{S}$):

```
1. $C \leftarrow \emptyset$.
2. While $E$ contains elements not covered by $C$:
    (a) Pick an element $e \in E$ not covered by $C$.
    (b) Add all sets $S_i$ containing $e$ to $C$.
```

To analyze Algorithm 3.1.1, we will need the following definition:

Definition 3.1.2 *A set E' of elements in E is* independent *if, for all $e_1, e_2 \in E'$, there is no $S_i \in C$ such that $e_1, e_2 \in S_i$.*

Now, we shall determine how strong an approximation Algorithm 3.1.1 is. Say that the *frequency* of an element is the number of sets that contain that element. Let F denote the maximum frequency across all elements. Thus, F is the largest number of sets from $\mathcal{S}$ that we might add to our cover C at any step in the algorithm. It is clear that the elements selected by the algorithm form an independent set, so the algorithm selects no more than $F|E'|$ elements, where E' is the set of elements picked in Step 2a. That is, ALG $\leq F|E'|$. Because every element is covered by some subset in an optimal set cover, we know that $|E'| \leq$ OPT for any independent set E'. Thus, ALG $\leq F\,$OPT, and Algorithm 3.1.1 is therefore an F–approximation.

Theorem 3.1.3 *Algorithm 3.1.1 is an F–approximation to Set Cover.*

Algorithm 3.1.1 is a good approximation if F is guaranteed to be small. In general, however, there could be some element contained in every set of $\mathcal{S}$, and Algorithm 3.1.1 would be a very poor approximation. So, we consider a different unweighted Set Cover approximation algorithm which uses the greedy strategy to yield a $\ln n$–approximation.

Algorithm 3.1.4 Set Cover(E, $\mathcal{S}$):

```
1. $C \leftarrow \emptyset$.
2. While $E$ contains elements not covered by $C$:
    (a) Find the set $S_i$ containing the greatest number of uncovered elements.
    (b) Add $S_i$ to $C$.
```

Theorem 3.1.5 *Algorithm 3.1.4 is a* $\ln \frac{n}{OPT}$*–approximation.*

Proof: Let $k = \text{OPT}$, and let E_t be the set of elements not yet covered after step i, with $E_0 = E$. OPT covers every E_t with no more than k sets. ALG always picks the largest set over E_t in step $t+1$. The size of this largest set must cover at least $|E_t|/k$ in E_t; if it covered fewer elements, no way of picking sets would be able to cover E_t in k sets, which contradicts the existence of OPT. So, $|E_{t+1}| \leq |E_t| - |E_t|/k$, and, inductively, $|E_t| \leq n\,(1-1/k)^t$.

When $|E_t| < 1$, we know we are done, so we solve for this t:

$$\begin{aligned}
&\left(1-\frac{1}{k}\right)^t < \frac{1}{n}\\
\Rightarrow\ & n < \left(\frac{k}{k-1}\right)^t\\
\Rightarrow\ & \ln n \leq t\ln\left(1+\frac{1}{k-1}\right) \approx \frac{t}{k}\\
\Rightarrow\ & t \leq k\ln n = \text{OPT}\ln n.
\end{aligned}$$

Algorithm 3.1.4 finishes within $\text{OPT}\ln n$ steps, so it uses no more than that many sets. We can get a better analysis for this approximation by considering when $|E_t| < k$, as follows:

$$\begin{aligned}
& n\left(1-\frac{1}{k}\right)^t = k\\
\Rightarrow\ & n\frac{1}{e^{t/k}} = k \text{ (because } (1-x)^{1/x} \leq \frac{1}{e} \text{ for all } x\text{).}\\
\Rightarrow\ & e^{t/k} = \frac{n}{k}\\
\Rightarrow\ & t = k\ln\frac{n}{k}.
\end{aligned}$$

Thus, after $k\ln\frac{n}{k}$ steps there remain only k elements. Each subsequent step removes at least one element, so $\text{ALG} \leq \text{OPT}\left(\ln\frac{n}{OPT}+1\right)$. ■

Theorem 3.1.6 *If all sets are of size* $\leq B$*, then there exists a* $(\ln B + 1)$*–approximation to unweighted Set Cover.*

Proof: If all sets have size no greater than B, then $k \geq n/B$. So, $B \geq n/k$, and Algorithm 3.1.4 gives a $(\ln B+1)$–approximation. ■

Now we extend Algorithm 3.1.4 to the weighted case. Here, instead of selecting sets by their number of uncovered elements, we select sets by the "efficiency" of their uncovered elements, or the number of uncovered elements *per unit weight*.

Algorithm 3.1.7 Weighted Set Cover($\mathcal{S}$, C, E, w):

```
1. $C \leftarrow \emptyset$, and $E' \leftarrow E$.
2. While $E$ contains uncovered elements:
    (a) $s \leftarrow \text{argmax}_{X \in \mathcal{S}} |X \cap E|/w(X)$.
    (b) $C \leftarrow C \cup s$, $\mathcal{S} \leftarrow \mathcal{S} \setminus \{s\}$, and $E' \leftarrow E' \setminus \mathcal{S}$.
```

Algorithm 3.1.7 was first analyzed in [5].

Theorem 3.1.8 *Algorithm 3.1.7 achieves a $\ln n$–approximation to Weighted Set Cover.*

Proof: For every picked set S_j, define θ_j as $|S_j \cap E|/w(S_j)$ at the time that S_j was picked. For each element e, let S'_j be the first picked set that covers it, and define $\text{cost}(e) = 1/\theta_j$. Notice that $\sum_{e \in E} \text{cost}(e) = \text{ALG}$.

Let us order the elments in the order that they were picked, breaking ties arbitrarily. At the time that the i^{th} element (call it e_i) was picked, E contained at least $n - i + 1$ elements. At that point, the "per-element" cost of OPT is at most $\text{OPT}/(n - i + 1)$. Thus, for at least one of the sets in OPT, we know that

$$\frac{|S \cap E|}{w(s)} \geq \frac{n - i + 1}{\text{OPT}}.$$

Therefore, for the set S_j picked by the algorithm, we have $\theta_j \geq (n - i + 1)/\text{OPT}$. So,

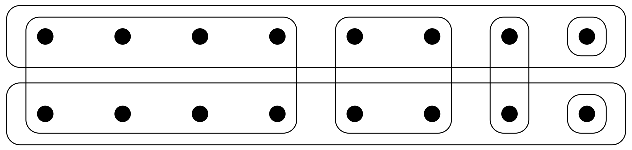
$$\text{cost}(e_i) \leq \frac{\text{OPT}}{n - i + 1}.$$

Over the execution of Algorithm 3.1.7, the value of i goes from n to 1. Thus, the total cost of each element that the algorithm removes is at most

$$\sum_{i=1}^{n} \frac{\text{OPT}}{n - i + 1} \leq \text{OPT} \ln n.$$

Thus, Algorithm 3.1.7 is a $\ln n$–approximation to Weighted Set Cover. ∎

The above analysis is tight, which we can see by the following example:



The dots are elements, and the loops represent the sets of S. Each set has weight 1. The optimal solution is to take the two long sets, with a total cost of 2. If Algorithm 3.1.7 instead selects the leftmost thick set at first, then it will take at least 5 sets. This example generalizes to a family of examples each with 2^k elements, and shows that no analysis of Algorithm 3.1.7 will make it better than a $O(\ln n)$–approximation.

A $\ln n$–approximation to Set Cover can also be obtained by other techniques, including LP-rounding. However, Feige showed that no improvement, even by a constant factor, is likely:

Theorem 3.1.9 *There is no $(1-\epsilon)\ln n$–approximation to Weighted Set Cover unless NP $\subseteq$ DTIME$(n^{\log\log n})$. [1]*

3.2 Min Makespan Scheduling

The *Min Makespan Problem* is: given n jobs to schedule on m machines, where job i has size s_i, schedule the jobs to minimize their makespan.

Definition 3.2.1 *The* makespan *of a schedule is the earliest time when all machines have stopped doing work.*

This problem is NP-hard, as can be seen by a reduction from Partition. The following algorithm due to Ron Graham yields a 2–approximation.

Algorithm 3.2.2 (Graham's List Scheduling) [2] Given a set of n jobs and a set of m empty machine queues,

1. Order the jobs arbitrarily.
2. Until the job list is empty, move the next job in the list to the end of the shortest machine queue.

Theorem 3.2.3 *Graham's List Scheduling is a 2–approximation.*

Proof: Let S_j denote the size of job j. Suppose job i is the last job to finish in a Graham's List schedule, and let t_i be the time it starts. When job i was placed, its queue was no longer than any other queue, so every queue is full until t_i. Thus, ALG $= S_i + t_i \le S_i + \frac{(\sum_{j=1}^n S_j)-S_i}{m} = \frac{1}{m}\sum_{j=1}^n S_j + (1-1/m)S_i$. It's easy to see that $S_i \le$ OPT and that $\frac{1}{m}\sum_{j=1}^n S_j \le OPT$. So, we conclude that ALG $\le (2-1/m)$OPT, which yields a 2–approximation. ■

This analysis is tight. Suppose that after the jobs are arbitrarily ordered, the job list contains $m(m-1)$ unit-length jobs, followed by one m-length job. The algorithm yields a schedule completing in $2m-1$ units while the optimal schedule has length m.

This algorithm can be improved. For example, by ordering the job list by increasing duration instead of arbitrarily, we get a (4/3)–approximation, a result proved in [3]. Also, this problem has a poly-time approximation scheme (PTAS), given in [4]. However, a notable property of Algorithm 3.2.2 is that it is an online algorithm, i.e., even if the jobs arrive one after another, and we have no information about what jobs may arrive in the furture, we can still use this algorithm to obtain a 2–approximation.

References

[1] Uriel Feige. A Threshold of $\ln n$ for Approximating Set Cover. In *J. ACM* 45(4), pp 634-652. (1998)

[2] Graham, R. Bounds for Certain Multiprocessing Anomalies. In *Bell System Tech. J.*, 45, pp 1563-1581. (1966)

[3] Ronald L. Graham. Bounds on Multiprocessing Timing Anomalies. In *SIAM Journal of Applied Mathematics*, 17(2), pp 416-429. (1969)

[4] Dorit S. Hochbaum, David B. Shmoys. A Polynomial Approximation Scheme for Scheduling on Uniform Processors: Using the Dual Approximation Approach. In *SIAM J. Comput.* 17(3), pp 539-551. (1988)

[5] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. In *Journal of Computer and System Sciences*, 9, pp 256-278. (1974) Preliminary version in *Proc. of the 5th Ann. ACM Symp. on Theory of Computing*, pp 36-49. (1973)