Lecture 6

Lecturer: Madhu Sudan *Scribe: Eric Lehman*

This lecture describes the set cover problem and introduces the multicut problem, which is discussed further in the next lecture.

1 The Set Cover Problem

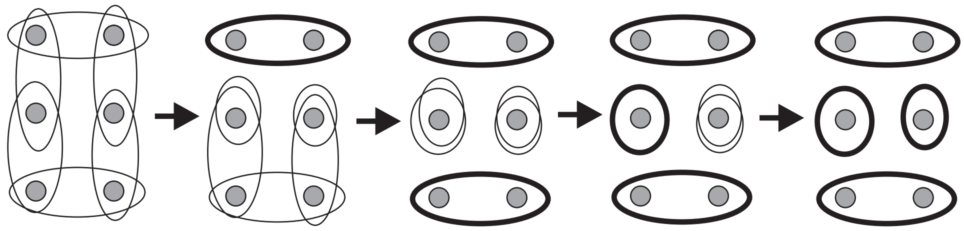
An instance of the set cover problem consists of a family of n subsets of $\{1, 2, \ldots, m\}$ whose union is all of $\{1, 2, \ldots, m\}$. The task is to find the smallest subfamily whose union is still all of $\{1, 2, \ldots, m\}$.

The set $\{1, 2, \ldots, m\}$ is called the *universe*. We denote the family of n subsets in a problem instance by $S_1, S_2, \ldots, S_n$. A subfamily *covers* the universe if its union is the entire universe.

This lecture examines the following greedy algorithm for the set cover problem. (The analysis is variously attributed to Chvatal 1979, Johnson 1974, and Lovasz 1975.)

- Add the largest set S_i to the cover.
- Delete all elements of S_i to obtain a smaller set cover problem.
- Repeat this procedure on the new instance.

An example run of the greedy algorithm is illustrated below. Selected sets are drawn with a dark boundary. The algorithm produces a cover of size 4, though covers of size 3 exist.



Here is a rough analysis of the greedy algorithm. Let $C(m, k)$ be the largest set cover produced by the greedy algorithm over all instances with universe of size at most m and minimum cover of size at most k. Since each set added by the greedy algorithm covers at least one additional element, $C(k, k) \leq k$. Furthermore, there must be some set of size at least m/k, and so the greedy algorithm always selects such a set. These observations give a recurrence:

$$\begin{aligned} C(k,k) &\leq k \\ C(m,k) &\leq 1 + C(m\,(1 - 1/k)\,, k) \end{aligned}$$

Consequently, $C(m, k) \leq k \log m$, meaning that the greedy algorithm is optimal to within a $\log m$ factor, where m is the size of the universe.

A primal-dual analysis gives a better approximation factor. To this end, we can recast the set cover problem as a linear program. With each set S_i, associate an indicator variable x_i. If $x_i = 1$, then set S_i is in the cover, and if $x_i = 0$, the set is not in the cover. We now get a relaxed linear program:

$$\min \sum_{i=1}^{n} x_i$$

$$\sum_{i \mid j \in S_i} x_i \geq 1 \quad \text{for all } j \in \{1, 2, \ldots, m\}$$

$$0 \leq x_i \leq 1 \quad \text{for all } i \in \{1, 2, \ldots, n\}$$

The first batch of contraints says that each element j in the universe must be contained by at least one set in the cover. The second batch is a relaxation of the contraint that each indicator variable x_i must be either 0 or 1.

The dual of set covering is a packing problem. Now each element j has a non-negative weight y_j. The objective is to make the sum of all element weights as large as possible, with the contraint that each set has capacity 1.

$$\max \sum_{j=1}^{m} y_j$$

$$\sum_{j \in S_i} y_j \leq 1 \quad \text{for all } i \in \{1, 2, \ldots, n\}$$

$$0 \leq y_j \quad \text{for all } j \in \{1, 2, \ldots, m\}$$

We can now track the behavior of the greedy algorithm in terms of both the primal and dual programs. We ensure that the dual solution is always feasible during a run of the algorithm. The algorithm terminates when the primal solution becomes feasible as well.

Initially, all the x_i and y_j variables are set to zero. Each time the greedy algorithm adds a set S_i to the cover, some of these variables are assigned non-zero values as follows:

primal: Assign $x_i = 1$.
dual: For all elements j in set S_i, assign $y_j = 1/(|S_i| \cdot B)$.

Each variable x_i is assigned a non-zero value at most once, since the corresponding set S_i is added to the cover at most once. Similarly, the algorithm specifies that after the set S_i is added to the cover, all elements in that set are deleted from the problem instance. As a consequence, each variable y_j is assigned a non-zero value exactly once over a run of the algorithm.

An important mystery variable, B, appears above in the assignment to the dual variables. Note that each time the greedy algorithm adds a set to the cover, the objective function in the primal is increased by 1, since we assign $x_i = 1$. However, the objective function in the dual is increased by only $1/B$, since we assign the value $1/(|S_i| \cdot B)$ to $|S_i|$ of the dual variables. Consequently, B is the ratio of the primal objective function to the dual objective function; in short, B is the approximation ratio we obtain for the greedy algorithm.

All that remains is to determine the smallest valid approximation ratio B. The limiting factor is that the dual solution must remain feasible over a run of the greedy algorithm. Recall that in the dual, elements are assigned weights, with the constraint that each set has capacity 1. In terms of variables, B must be large enough so that when we assign elements weights like $1/(|S_i| \cdot B)$, the capacity contraint $\sum_{j \in S_i} y_j \leq 1$ remains satisfied for every set S_i.

Claim 1 *After a run of the greedy algorithm, for every set S_i,*

$$\begin{aligned} \sum_{j \in S_i} y_j &\leq \frac{1}{B} + \frac{1}{2B} + \frac{1}{3B} + \ldots + \frac{1}{|S_i| \cdot B} \\ &= \frac{H_{|S_i|}}{B}. \end{aligned}$$

(Here, H_k denotes the k-th harmonic number, which is $1 + 1/2 + 1/3 + \ldots + 1/k \sim \log k$.)

This claim implies that all capacity constraints are satisfied if we choose $B = H_{m'}$ where m' is the size of the largest of the sets $S_1, S_2, \ldots, S_n$. This implies an $H_{m'} \sim \log m'$ approximation ratio. Thus we have improved the approximation ratio from the log of the size of the universe to the log of the size of the largest set.

Proof We show that the k^{th}-to-last element of S_i to be assigned a weight gets weight at most $1/kB$. The claim follows. Suppose k elements of set S_i have not yet been assigned a weight. The greedy algorithm selects the largest set, which is at least as large as S_i, which has k elements remaining. Consequently, the next weight assigned can be no greater than $1/kB$. ■

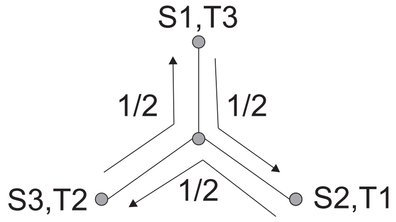
2 Introduction to the Multicut Problem

An instance of the classic $s-t$ *min-cut problem* consists of a graph $G = (V, E)$ together with two distinct vertices $s, t \in V$. The objective is to separate s from t by cutting the minimum number of edges. That is, one must find a subset of edges F such that s and t lie in distinct connected components of the graph $G' = (V, E \setminus F)$. This problem can be solved in polynomial time by using a maximum $s-t$ flow algorithm and relying on the maxflow mincut theorem. Recently, David Karger found faster algorithms that do not rely on flow algorithms.

An instance of the more general *multiway cut problem* consists of a graph together with distinct vertices $s_1, s_2, \ldots, s_k$. Now the task is to separate s_i from s_j for all $i \neq j$ by cutting the minimum number of edges. For this problem, there is a 2-approximation algorithm due to Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis in 1992.

Generalizing further, we reach the multicut problem. Here an instance consists of a graph and a set of pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. The objective is to separate s_i from t_i for all i by cutting the minimum number of edges.

The minimum multicut problem is closely related to a maximum multicommodity flow problem, much as $s-t$ mincuts are related to $s-t$ maxflows. In the multicommodity flow problem, there is a graph with source-sink pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$. The objective is to route the maximum flow from the sources to the corresponding sinks, with the contraint that each edge can carry at most one unit of flow of all commodities combined. Unlike the single commodity case, the minimum multicut is not always equal to the maximum multicommodity flow, as the example below shows.



Here, the maximum flow is $\frac{3}{2}$, but the minimum multicut is 2. Limits on the difference between these two quantities were first found by Leighton and Rao in 1988. In 1996, Garg, Vazirani, and Yannakakis showed that:

$$\text{max flow} \;\leq\; \text{min } k\text{-multicut} \;\leq\; \text{max flow} \cdot \Theta(\log k).$$

Their approach to constructing a multicut satisfying these inequalities is to assign a length to each edge by solving a relaxed linear program. They then grow a ball around each source, ensuring that no ball becomes large enough to wholly contain a source-sink pair. They then take all edges that cross the boundary of a ball as the multicut. Details in next lecture.