6.891 Approximation Algorithms February 4, 2000

Lecture 1

Lecturer: David P. Williamson *Scribe: John Dunagan*

1 An Introduction to Approximation Algorithms

How can we solve NP-hard optimization problems efficiently (assuming $P \neq NP$)? Chvátal told a recent joke on this subject at SODA which we repeat here. In Communist Eastern Block countries in the 60s and 70s, it was possible to be intelligent, honest, or a member of the communist party, but no more than two of the three at once. Algorithms for NP-hard optimization problems are like this in the sense that we would like them to:

(i) find optimum solutions

(ii) in polynomial time

(iii) for any instance

However, if $P \neq NP$, no algorithm meets more than two out of the three requirements.

If we drop requirement (iii), then we are lead to the study of special cases. If we drop requirement (ii), then we are lead to the field of integer programming, and the many techniques there, such as branch-and-bound, branch-and-cut, etc. If we drop requirement (i), then we are lead to heuristics, which includes greedy algorithms, local search algorithms (including the more sophisticated variants such as simulated annealing and genetic algorithms), and approximation algorithms. This class will focus on approximation algorithms.

Definition 1 *An algorithm is an* α-approximation algorithm *for an optimization problem* Π *if*

1. *The algorithm runs in polynomial time*

2. *The algorithm always produces a solution which is within a factor of* α *of the value of the optimal solution*

The factor α is known as the "performance guarantee" of the algorithm.

Throughout the course we will use the following convention: for minimization problems, $\alpha > 1$ (this is universal), while for maximization problems, $\alpha < 1$ (this is not universal). In the literature, $1/\alpha$ is sometimes used for maximization problems.

Why do we study approximation algorithms?

1. NP-hard problems exist and need solutions.
2. As ideas for algorithms that are used in practice for item #1.
3. As a mathematically rigorous way of studying heuristics.
4. Because it's fun!
5. Because it tells us how hard problems are.

Let us briefly touch on item 5 above, beginning with another definition:

Definition 2 *A* polynomial-time approximation scheme (PTAS) *for a minimization problem is a family of algorithms* $\{A_\epsilon : \epsilon > 0\}$ *such that for each* $\epsilon > 0$, A_ϵ *is a* $(1+\epsilon)$*-approximation algorithm which runs in time polynomial in the input size for fixed* ϵ. *For a maximization problem, we require that* A_ϵ *is a* $(1-\epsilon)$*-approximation algorithm.*

Some problems for which *PTAS*'s exist are knapsack, Euclidean TSP (Arora 1996, Mitchell 1996), and many scheduling problems. Other problems like MAX SAT, MAX CUT and Metric TSP are harder. They all belong to the class of MAX SNP-hard problems, which are characterized by the following theorem:

Theorem 1 *(Arora, Lund, Motwani, Sudan, Szegedy 1992) There does not exist a* PTAS *for any* MAX SNP-hard *problem unless* $P = NP$.

There is a similar, but even more discouraging result with respect to *MAX CLIQUE*:

Theorem 2 *(Håstad 1996) There does not exist a* $O(n^{1-\epsilon})$ *approximation algorithm for any* $\epsilon > 0$ *for* MAX CLIQUE *unless* $NP \subseteq RP$.

What is *MAX CLIQUE*? Given a graph $G = (V, E)$, find the clique $S \subset V$ of maximum size $|S|$. And what is a clique?

Definition 3 *A* clique *S is a set of vertices for which each vertex pair has its corresponding edge included (that is,* $i \in S$, $j \in S$ *implies* $(i, j) \in E$*).*

Note that there is a trivial approximation algorithm for *MAX CLIQUE* with performance guarantee $n = |V|$. Simply take a single vertex; this is trivially a clique. The size of any clique cannot be more than n, so the algorithm has a performance guarantee of $n/1 = n$. Håstad's result tells us that we do not expect to do much better than this trivial algorithm.

This brief survey is meant to convey that some NP-hard optimization problems are much harder than others. While some problems are approximable to within any factor you want, other problems are not approximable beyond a trivial amount.

The previous results about MAX CLIQUE and MAX SNP come from the theory of probabilistically checkable proofs.

The theory of approximation algorithms is a set of positive results showing that some problems are efficiently approximable. The theory of probabilistically checkable proofs (PCP) is a set of negative results showing that many problems cannot be approximated better than a certain factor. Together the two form a complete theory of NP-hard optimization problems. In this class we will only mention PCP results; our focus will be on deriving approximation techniques. This will be done by studying a central body of techniques, mostly from the 90's, that will enable you to go forth and design your own approximation algorithms.

We now post two central theses of this course:

- There is a central core of techniques for designing approximation algorithms. Know these, and you know a lot.
- Linear Programming and Integer Programming are central tools for designing and analyzing approximation algorithms.

Both of these theses will be illustrated in our discussion of Set Cover, an NP-hard optimization problem.

2 Set Cover

Weighted Set Cover (SC)

- **Input:**
  - Ground elements $E = \{e_1, e_2, \cdots, e_n\}$
  - Subsets $S_1, S_2, \cdots, S_m \subseteq E$
  - Weights $w_1, w_2, \cdots, w_m \geq 0$
- **Goal:** Find a set $I \subseteq \{1, 2, \cdots, m\}$ that minimizes $\sum_{i \in I} w_i$, such that $\bigcup_{i \in I} S_i = E$.

For the *unweighted SC* problem, we take $w_j = 1$ for all j.

Why should we care about the Set Cover Problem? First, the problem shows up in various applications. A colleague of Dr. Williamson at IBM applied the set cover problem to try to find relevant features of boot sector viruses for the IBM product IBM AntiVirus.

- *Elements*: Known boot-sector viruses (about 150 of them).

- *Sets*: 3 byte sequences in viral boot sectors not found in "good" code (about 21,000 of them).

Once a relevant set cover was found, the 3 byte sequences were used as "features" for a neural classifier that would determine when a boot sector virus was present or not. Since many boot sector viruses are written by modifying previous ones it was hoped that this would allow detection of previously unknown boot sector viruses. A small set cover (much less than 150 sets) was desired in order to avoid overfitting the data. Using the greedy algorithm discussed later in the lecture, a set cover of 50 sets was found (actually, the cover was such that each element was contained in four distinct sets of the solution). The neural classifier successfully found many unknown boot sector viruses in practice.

A second reason to care about Set Cover is that it generalizes other problems. Consider the following problem:

Weighted Vertex Cover (VC)

- **Input:**
  - An undirected graph $G = (V, E)$
  - Weights $w_i \geq 0 \ \forall i \in V$
- **Goal:** Find a set C that minimizes $\sum_{i \in C} w_i$, such that for every $(i, j) \in E$, we have either $i \in C$ or $j \in C$.

To see that VC is a special case of SC, consider the following identification:

- *Elements in SC*: all edges in G (the VC edges).
- *Sets in SC*: $S_i = \{\text{all edges incident to vertex } i\}$.

2.1 An Integer Programming Formulation

Now we progress to the second thesis. Let's write Set Cover as an integer program. Here, we create a variable x_j for each subset S_j. If $j \in I$, then $x_j = 1$, otherwise $x_j = 0$.

$$\text{Min} \quad \sum_{j=1}^{m} w_j x_j$$

subject to:

$$\sum_{j : e_i \in S_j} x_j \geq 1 \qquad \forall e_i \in E$$

$$x_j \in \{0, 1\}$$

The inequality constraints just state that for each item in our ground set, we must choose at least one set that covers that item. This can't be solved in polynomial time, so we relax the integrality requirement. The resulting linear program (LP) can be solved in polynomial time.

$$
\begin{aligned}
&\text{Min} \quad \sum_{j=1}^{m} w_j x_j \\
&\text{subject to:} \\
&\qquad \sum_{j:e_i \in S_j} x_j \geq 1 \qquad\qquad \forall e_i \in E \\
&\qquad x_j \geq 0
\end{aligned}
$$

If you are not familiar with any of the myriad ways to solve LP's in polynoimal time, don't worry. We will just be using polynomial time LP solvers as a black box. We close our eyes and the LP solver returns a solution x^*.

Let OPT be the optimal objective value for the integer program, and let Z^*_{LP} be the optimal objective value for the linear program. Then $Z^*_{LP} \leq OPT$ since the solution space for the integer program is a subset of the solution space of the linear program. We can rewrite this as

$$\sum_{j=1}^{m} w_j x^*_j \leq OPT.$$

We now proceed to our first approximation algorithm.

2.2 Method I: Rounding

We define $f_i = |\{j : e_i \in S_j\}|$ and $f = \max_i f_i$. Our first algorithmic idea is just to round up any variable x^*_j that is sufficiently large. Let

$$I = \{j | x^*_j \geq \frac{1}{f}\}.$$

Lemma 3 *I is a set cover.*

Proof: Suppose there is an element e_i such that $e_i \notin \bigcup_{j \in I} S_j$. Then for each set S_j of which e_i is a member, we have $x^*_j < 1/f$. So

$$
\begin{aligned}
\sum_{j:e_i \in S_j} x^*_j &< \frac{1}{f} \cdot |\{j : e_i \in S_j\}| \\
&\leq 1,
\end{aligned}
$$

since $|\{j : e_i \in S_j\}| \leq f$. But this violates the linear programming constraint for e_i. $\square$

Theorem 4 *(Hochbaum '82) Rounding is an f-approximation algorithm for set cover.*

Proof: It is clear that the rounding algorithm is a polytime algorithm. Furthermore,

$$\begin{aligned}\sum_{j \in I} w_j &\leq \sum_{j=1}^{n} w_j (f \cdot x_j^*) \\ &= f \sum_j w_j x_j^* \\ &\leq f \cdot OPT.\end{aligned}$$

The first inequality follows since $j \in I$ only if $x_j^* f \geq 1$. □

Now we consider a second rounding technique. If you are unfamiliar with duals, the course handouts should help.

2.3 Method II: Dual Rounding

Another way to use the rounding method is to apply it to the dual solution. The dual of the linear programming relaxation for set cover is:

$$\begin{aligned}&\text{Max} \quad \sum_i y_i \\ &\text{subject to:} \\ &\qquad \sum_{i:e_i \in S_j} y_i \leq w_j \qquad\qquad \forall S_j \\ &\qquad y_i \geq 0 \qquad\qquad \forall e_i \in E.\end{aligned}$$

If we have a feasible dual solution y, then

$$\sum_i y_i \leq Z_{LP}^* \leq OPT$$

by weak duality. Taking advantage of the fact that finding a dual solution is just as easy as finding a primal solution, we present the following algorithm. This time, we take all sets whose corresponding dual inequality is *tight*; that is, it is met with equality. We let our solution I' be

$$I' = \{j | \sum_{i, e_i \in S_j} y_i^* = w_j\}.$$

Lemma 5 *I' is a set cover.*

Proof: Suppose $\exists e_i \notin \bigcup_{j \in I'} S_j$. Then for each S_j containing e_i

$$\sum_{i:e_i \in S_j} y_i^* < w_j,$$

so we can increase y_i^* by some positive amount and remain feasible, which contradicts the optimality of y^*. Just to make the positive amount explicit, it's given by

$$\epsilon = \min_{j:e_i \in S_j} \{w_j - \sum_{k:e_k \in S_j} y_k^*\}$$

$\square$

Theorem 6 *(Hochbaum '82) Dual rounding is an f-approximation algorithm.*

Proof: Because we choose set S_j only if its constraint is tight, we have

$$\begin{aligned}
\sum_{j \in I'} w_j &= \sum_{j \in I'} \sum_{i:e_i \in S_j} y_i^* \\
&= \sum_i y_i^* |\{j \in I' : e_i \in S_j\}| \\
&\leq f \sum_i y_i^* \\
&\leq f \cdot OPT.
\end{aligned}$$

$\square$

Maria Minkoff observed in class that complementary slackness (a property of LP's explained in the course handouts) guarantees that whenever the Rounding algorithm includes a set S_j (because $x_j^* \geq \frac{1}{f}$), the corresponding dual constraint is tight, so Dual rounding also includes the set S_j in its solution. Thus, Dual-Rounding never obtains a better solution than Rounding ($I \subseteq I'$).

Now we are going to present an algorithm which "uses LP" in the sense that its design and analysis are substantially motivated by our understanding of LP's, but which never needs to call an LP solver. This leads to an approximation algorithm with the same performance guarantee as before, but with a significantly improved running time.

2.4 Method III: Primal-Dual

The following algorithm behaves much like Dual rounding above, except that it constructs its own dual solution, rather than finding the optimal dual LP solution. The motivation for this is that we only needed three facts in our above analysis. We needed that y^* was feasible, that every set included in our solution corresponded to a tight dual constraint, and that our solution was a set cover.

In fact, Lemma 5 gives us exactly the idea we need on how to construct a dual solution. Let I' contain all the indices of the sets whose dual inequality is currently tight. Then the lemma shows that if I' is not a set cover, there is a way to improve the overall value of the dual solution, and create a new tight constraint for some set that contains an item not currently covered.

We formalize the algorithm below. In the following algorithm, let $\tilde{y}$ represent the dual solution that we iteratively construct. During the entire algorithm, we maintain that $\tilde{y}$ is a dual feasible solution.

Primal-Dual

$I \leftarrow \emptyset$
$\tilde{y}_i \leftarrow 0 \;\; \forall i$
while $\exists e_k : e_k \notin \bigcup_{j \in I} S_j$
    $l = \arg\min_{j : e_k \in S_j} \left\{ w_j - \sum_{i : e_i \in S_j} \tilde{y}_i \right\}$
    $\epsilon_l \leftarrow w_l - \sum_{i : e_i \in S_l} \tilde{y}_i$
    $\tilde{y}_k \leftarrow \tilde{y}_k + \epsilon_l$
    $I \leftarrow I \cup \{l\}$.

Note that the function arg min returns the argument (index, in this case) that minimizes the expression. It is not hard to give this algorithm in words: at each step, we look for an element e_k that is not covered. For that element, we consider all the sets that include the element e_k, and their corresponding dual constraints. Whichever dual constraint is closest to being tight, we increase the dual variable y_k^* until the constraint is tight (satisfies the inequality as an equality) and include the same set (corresponding to the dual constraint) in our set cover. Now the analysis is nearly identical to that for Dual-Rounding.

Lemma 7 *Primal-Dual returns a set cover.*

Proof: This is the termination condition for the while loop. □

Lemma 8 *Primal-Dual constructs a dual feasible solution.*

Proof: We proceed by induction on the loops of the algorithm. The base case is trivial since initially

$$\sum_{i : e_i \in S_j} \tilde{y}_i = 0 \leq w_j \;\; \forall j$$

For the inductive step, assume that upon entering an iteration of the while loop we have

$$\sum_{i : e_i \in S_j} \tilde{y}_i \leq w_j \;\; \forall j$$

The only dual variable value changed by the while loop is $\tilde{y}_k$, so the inequalities for S_j where $e_k \notin S_j$ are unaffected. If $e_k \in S_j$, then by our choice of l

$$\begin{aligned} \sum_{i : e_i \in S_j} \tilde{y}_i + \epsilon_l &= \sum_{i : e_i \in S_j} \tilde{y}_i + (w_l - \sum_{i : e_i \in S_l} \tilde{y}_i) \\ &\leq \sum_{i : e_i \in S_j} \tilde{y}_i + (w_j - \sum_{i : e_i \in S_j} \tilde{y}_i) \\ &\leq w_j. \end{aligned}$$

□

Lemma 9 *If $j \in I$ then $\sum_{i:e_i \in S_j} \tilde{y}_i = w_j$.*

Proof: In the step where set j was added to I, we increased $\tilde{y}_k$ by exactly enough to make constraint j tight. □

The rest of the proof is essentially the same as Theorem 6 in the case of dual rounding.

Theorem 10 *(Bar-Yehuda, Even '81[1]) Primal-Dual is an f-approximation algorithm for the set cover problem.*

Proof:

$$\begin{aligned}\sum_{j \in I} w_j &= \sum_{j \in I} \sum_{i:e_i \in S_j} \tilde{y}_i \\ &\leq \sum_{i=1}^{n} \tilde{y}_i |\{j : e_i \in S_j\}| \\ &\leq f \cdot \sum_i \tilde{y}_i \\ &\leq f \cdot OPT.\end{aligned}$$

The first equality follows from Lemma 9. The next inequality follows since each $\tilde{y}_i$ can appear in the double sum at most $|\{j : e_i \in S_j\}|$ times. The next inequality follows by the definition of f, and the last inequality follows from weak duality. □

It is worth noting that the observation by Maria Minkoff no longer holds; since we are not constructing an optimal dual solution, we cannot apply the complementary slackness analysis.

2.5 Method IV: Greedy Algorithm

So far every technique we have tried has led to the same result: an f-approximation algorithm for set cover. In the case of set cover, using a natural greedy heuristic yields an improved approximation algorithm.

The intuition here is straightforward. At each step choose the set that gives the most "bang for the buck." That is, select the set that minimizes the cost per additional element covered.

Before examining the algorithm, we define two more quantities:

$$H_n \equiv 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \approx \ln n$$

$$g \equiv \max_j |S_j|.$$

[1] although the date is before '82, this algorithm followed Hochbaum's work

Greedy

$I \leftarrow \emptyset$
$\tilde{S}_j \leftarrow S_j \;\; \forall j$
while $\bigcup_{j \in I} S_j \neq E$
  $l \leftarrow \arg\min_{j:\tilde{S}_j \neq \emptyset} \frac{w_j}{|\tilde{S}_j|}$
  $I \leftarrow I \cup \{l\}$
  $\tilde{y}_i \leftarrow \frac{w_l}{|\tilde{S}_l| H_g} \;\; \forall e_i \in \tilde{S}_l$ (†)
  $\tilde{S}_j \leftarrow \tilde{S}_j - S_l \;\; \forall j$

Note: The † step has been added only to aid the proof and is not actually part of the algorithm.

The last step is simply meant to explain how we update the universe to account for our previous action. After we include a new set, we subtract out of the universe all the elements covered by that set. We now make the following shocking claim.

Claim 11 *Greedy is an H_g approximation algorithm.*

In contrast to the primal-dual algorithm, we are no longer using the dual variables to guide our solution. Instead, we are simply using them for purposes of analysis. The step where we assign $\tilde{y}_i$ defines the dual solution we construct.

First we note that $\tilde{y}_i$ is set only once for each e_i. It is set when that element is included and not touched again.

We now assume for the moment that $\tilde{y}$ is a dual feasible solution. Given this, we can prove the claim.

Theorem 12 *Greedy is an H_g approximation algorithm.*

Proof: The cost of the solution is

$$\sum_{j \in I} w_j = H_g \sum_{i=1}^{n} \tilde{y}_i \leq H_g \cdot OPT.$$

The first equality follows from the observation that when j is added to I,

$$w_j = H_g \sum_{i:e_i \in \tilde{S}_j} \tilde{y}_i$$

where $\tilde{S}_j$ is the set S_j with all the elements that have been covered in previous steps removed. This is simply by the definition of the dual variables as they are set in step (†). The second inequality is just weak duality. □

We now need to prove that the solution is dual feasible.

Lemma 13 *The solution $\tilde{y}$ is dual feasible.*

Proof: Pick an arbitrary S_j. We will show that

$$\sum_{i:e_i \in S_j} \tilde{y}_i \leq w_j$$

.

Let a_k be the number of uncovered elements in S_j at beginning of k^{th} iteration of the algorithm, and let $l \equiv$ total number of iterations. Then $a_1 = |S_j|$ and $a_{l+1} = 0$. Finally, let A_k be the set of previously uncovered elements of S_j covered in iteration k. We immediately find that $|A_k| = a_k - a_{k+1}$.

Suppose that set S_r is chosen in the k^{th} iteration. Then, we have that for every element e_i (and its corresponding dual variable $\tilde{y}_i$),

$$\tilde{y}_i = \frac{w_r}{H_g |S_r|} \leq \frac{w_j}{H_g a_k}$$

The equality is by definition on $\tilde{y}_i$, and the inequality follows since S_r minimizes the ratio $\frac{w_r}{S_r}$.

We now finish proving the lemma as follows.

$$\begin{aligned}
\sum_{i:e_i \in S_i} \tilde{y}_i &= \sum_{k=1}^{l} \sum_{i:e_i \in A_k} \tilde{y}_i \\
&\leq \sum_{k=1}^{l} (a_k - a_{k+1}) \cdot \frac{w_j}{H_g a_k} \\
&= \frac{w_j}{H_g} \sum_{k=1}^{l} \frac{a_k - a_{k+1}}{a_k} \\
&\leq \frac{w_j}{H_g} \sum_{k=1}^{l} \left(\frac{1}{a_k} + \frac{1}{a_k - 1} + \cdots + \frac{1}{a_{k+1} + 1} \right) \\
&\leq \frac{w_j}{H_g} \sum_{i=1}^{a_1} \frac{1}{i} \leq w_j.
\end{aligned}$$

In the first line, we simply note that the sum of all the $\tilde{y}_i$ variables is the sum over all iterations of the y_i variables set in a particular iteration. In the second line, we use our identity for the number of elements in A_k, and we also plug in our bound on $\tilde{y}_i$. The rest is just algebraic manipulation. □

The following complexity results give a sense of how good our approximation algorithms are in relation to what is possible.

Theorem 14 *(Lund, Yannkakis '92, Feige '96, Raz, Safra '97, Arora, Sudan '97) Let $n = |T| =$ the size of the ground set. Then:*

- *If there exists a $c \ln n$-approximation algorithm where $c < 1$ then $NP \subseteq DTIME(n^{O(\log^k n)})$*

- *There exists some* $0 < c < 1$ *such that if there exists a* $c \log n$*-approximation algorithm for set cover, then* $P = NP$.

So in one direction, our approximation seems to be best possible. However, f and H_g are not strictly comparable measures. f measures the maximum number of times an element appears in different sets, while g measures the size of the largest set. Although f can be much greater than $\log n$, and thus H_g is a much better approximation ratio in general, we do not have a matching PCP bound for the case of constant f.

For example, the Vertex Cover problem corresponds to the case of $f = 2$. In this case, there is no known algorithm that achieves better than a 2-approximation, but the strongest inapproximability result is

Theorem 15 *(Håstad '97) If there exists an* α*-approximation algorithm for vertex cover with* $\alpha < \frac{7}{6}$ *then* $P = NP$.

So even for this extremely simple case we do not know everything there is to know.