CSE347 Lecture 23

Approximation Algorithms I: Greedy & Random

1. Set cover
2. MAX 3-SAT

Last time, we saw something very interesting: our on-line algorithm for load balancing obtained a makespan that was at most twice the optimum, even though it's NP-complete to determine the optimum, even if we knew the full sequence of requests. That is, we see that it is possible to solve at least one NP-complete optimization problem, so long as we tolerate solutions that are not quite optimal, but are "close" in some sense. Actually, for most NP-complete optimization problems, there is some $\alpha(n)$ such that on input x of size n, if the optimum solution is some value $OPT(x)$, then we can design a polynomial time algorithm that obtains a solution to x of value at least/most $\alpha(n)OPT(x)$ (as appropriate for maximization/minimization objectives). This is an $\alpha(n)$-approximation algorithm for the problem. So, taking the entire set of jobs and running our online algorithm on them in arbitrary order is a "2-approximation algorithm" for (offline) load balancing. (In addition to being 2-competitive for the on-line problem.) The approximation factors $\alpha(n)$ that we obtain may or may not be acceptably small, but unlike

average-case analysis, this is often a way of handling NP-complete problems. Indeed, the vast majority of research on algorithms over the last few decades has focused primarily on the design and analysis of such approximation algorithms.

Set Cover

Our 2-approximation algorithm for load balancing was a simple greedy algorithm. It turns out that greedy algorithms often provide reasonable approximate solutions. We'll see this for a very general problem, Set Cover: for a "universe" set U of size n, we are given a list of subsets, $S_1, S_2, \ldots, S_m \subseteq U$. We'll further associate weights, $w_1, w_2, \ldots, w_m \geq 0$ with the sets. Our goal is to find a collection $C \subseteq \{1, 2, \ldots, m\}$ that "covers" U, i.e. $\bigcup_{i \in C} S_i = U$, of minimum total weight $\sum_{i \in C} w_i$.

Set Cover generalizes Vertex Cover. How? Given $G=(V,E)$, take $U=E$, and for each $v \in V$, define S_v to be the set of edges with v as an endpoint, and put $w_v = 1$ for all $v \in V$. Then we are minimizing the # of sets (= vertices) to cover all of the edges. So, Set Cover is also NP-complete, and clearly quite versatile. We'll give a $O(\log n)$-approximation for Set Cover, and hence, also for any problem it can express easily.

A natural greedy objective for Set Cover is to maximize the # of elements we cover per unit of weight: if R is

the set of remaining elements to cover. we choose the set S_i to minimize $\frac{w_i}{|S_i \cap R|}$. We then update $R \leftarrow R - S_i$, and repeat until $R = \emptyset$.

It's immediate that this finds a set cover solution. Towards analyzing the algorithm, we define $c_u = \frac{w_i}{|S_i \cap R|}$ to be the cost paid to cover each element $u \in S_i \cap R$ when we choose S_i. Notice, the total cost we pay satisfies

$$\sum_{u \in U} c_u = \sum_{i \in \mathcal{I}} \sum_{\substack{u \in S_i \\ u \notin S_j \, j<i}} \frac{w_i}{|S_i - \bigcup_{j<i} S_j|} = \sum_{i \in \mathcal{I}} |S_i - \bigcup_{j<i} S_j| \frac{w_i}{|S_i - \bigcup_{j<i} S_j|} = \sum_{i \in \mathcal{I}} w_i$$

To compare this to the optimum solution, we'll relate the cost we pay to cover an arbitrary set S_k (that may not be part of our collection $\mathcal{I}$) to its weight w_k. If we can show $\sum_{u \in S_k} c_u \leq \alpha \cdot w_k$, then we're doing at most α times worse than simply picking S_k. And indeed,

Lemma: For every S_k, $\sum_{u \in S_k} c_u \leq H_{|S_k|} \cdot w_k$. (Again, $H_{|S_k|} = \sum_{\ell=1}^{|S_k|} \frac{1}{\ell}$)

Proof: Let s_j be the j^{th} element of S_k covered by our algorithm, so $S_k = \{s_1, \dots, s_d\}$ where $d = |S_k|$ (we break ties arbitrarily). Now, consider the iteration in which we cover s_j. By definition, $s_j, s_{j+1}, \dots, s_d$ were not yet covered, so $|S_k \cap R| \geq d - j + 1$ (note, s_j might be covered in the same iteration as $s_{j-1}, s_{j-2}, \dots$, so $|S_k \cap R|$ could be larger). Now, our algorithm chooses some S_i to minimize $\frac{w_i}{|S_i \cap R|}$, and $c_{s_j} = \frac{w_i}{|S_i \cap R|}$. Thus, $c_{s_j} \leq \frac{w_k}{|S_k \cap R|} \leq \frac{w_k}{d-j+1}$. So, the total cost we pay for S_k satisfies $\sum_{s_j \in S_k} c_{s_j} \leq \sum_{j=1}^{d} \frac{w_k}{d-j+1} = \sum_{\ell=1}^{d} \frac{w_k}{\ell} = w_k \cdot H_d$ ▨

Theorem: For $d^* = \max_i |S_i|$, our algorithm is a H_{d^*}-approximation

algorithm for Set Cover.

Proof: Let $\mathcal{C}^*$ be a collection of minimum total weight w^*. That is, $w^* = \sum_{i \in \mathcal{C}^*} w_i$. By our lemma, each $w_i \geq \frac{1}{H_{|S_i|}} \sum_{u \in S_i} c_u$ where $H_{|S_i|} \leq H_{d^*}$. Now, since $\mathcal{C}^*$ is a set cover, $\sum_{i \in \mathcal{C}^*} \sum_{u \in S_i} c_u \geq \sum_{u \in U} c_u$.

Thus: $w^* \geq \sum_{i \in \mathcal{C}^*} \frac{1}{H_{d^*}} \sum_{u \in S_i} c_u \geq \frac{1}{H_{d^*}} \sum_{u \in U} c_u = \frac{1}{H_{d^*}} \sum_{i \in \mathcal{C}} w_i$ ▨

In particular, since no set contains more than n elements, this is a $H_n \leq (\log_2 n + 1)$-approximation. (But if the sets are all small, it is much better...)

MAX 3-SAT

Another simple technique that provides reasonable approximate solutions surprisingly often is to pick a solution at random. We can illustrate this with an optimization version of 3-SAT: recall that in 3-SAT, we were given a 3-CNF formula: an AND of ORs of at most three literals, Boolean variables or their negations. We wanted to know if there was an assignment that satisfied all m clauses. But, more generally, we might want to satisfy as many of these constraints as possible, for example, if they can't all be satisfied. This is the MAX 3-SAT problem; actually, today we'll focus on the variant of the problem in which every clause has exactly three literals on distinct Boolean variables. It turns out that it's possible to add constraints of this form can be used to enforce that "dummy variables" z_1, z_2, z_3

can only be set to false in any satisfying assignment. Then we can pad out small clauses, e.g., x or $x \vee y$, to $x \vee z_1 \vee z_2$ or $x \vee y \vee z_1$, respectively. So this problem is still NP-complete.

Now, as promised: we'll be able to find a good assignment by, surprisingly, assigning each x_i independently and uniformly at random. That's the algorithm.

Let's fix some clause $C_j = \ell_1 \vee \ell_2 \vee \ell_3$ of our formula. What is the probability that C_j is satisfied by our assignment? C_j is only false when each of ℓ_1, ℓ_2, and ℓ_3 is simultaneously false. This occurs with probability $\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2} = \frac{1}{8}$. So, C_j is satisfied with probability $\frac{7}{8}$. Now we know that the indicator random variable $I_j = \begin{cases} 1 \text{ if } C_j \text{ satisfied} \\ 0 \text{ otherwise} \end{cases}$ has expected value $\mathbb{E}[I_j] = \Pr[C_j \text{ satisfied}] = \frac{7}{8}$. Moreover, the expected number of satisfied clauses is $\mathbb{E}\left[\sum_{j=1}^{m} I_j\right]$ $= \sum_{j=1}^{m} \mathbb{E}[I_j]$ by linearity of expectation. But, by our calculation we find $\mathbb{E}\left[\sum_{j=1}^{m} I_j\right] = m \cdot \frac{7}{8}$. So, in expectation $\frac{7}{8}$ of the clauses are satisfied. It follows that there must be an assignment that satisfies $\frac{7}{8}$ of the clauses (or else the average would be lower). Since no assignment can satisfy more than all m clauses, an assignment that satisfies $\frac{7}{8}m$ clauses is a $\frac{7}{8}$-approximation to the optimum.

Theorem A random assignment satisfies $\frac{7}{8}m$ clauses with probability at least $\frac{1}{8m}$.

As before, we can amplify our probability of success by repeating $8m \log\frac{1}{\delta}$ times, and taking the assignment that satisfies the most clauses. Alternatively, we can repeat until we find one that satisfies $\frac{7}{8}m$ clauses, which takes $8m$ tries in expectation.

Proof: Let p be the probability that a random assignment satisfies at least $\frac{7}{8}m$ clauses. Let m' be the largest # of clauses that is less than $\frac{7}{8}m$. Then we can bound the expected # of clauses as: $\mathbb{E}[\#\text{ sat. clauses}] = \mathbb{E}[\#\text{sat} \mid \geq \frac{7}{8}m \text{ sat}]\, p + \mathbb{E}[\#\text{sat} \mid < \frac{7}{8}m \text{ sat.}]\,(1-p)$

≤ 1

So $\frac{7}{8}m \leq m \cdot p + m'(1-p)$. But, notice, m' is the largest integer less than $\frac{7}{8}m$, so $\frac{7m-1}{8} \geq m'$ (since $8m'$ is the largest mult. of 8 $< 7m$ we get $m' = \frac{\text{largest mult. of 8} < 7m}{8} \leq \frac{7m-1}{8}$). Therefore, $\frac{1}{8} \leq \frac{7}{8}m - m' \leq m \cdot p$, so $p \geq \frac{1}{8m}$ as needed. ▨

Another surprising fact is that this is the best possible approximation ratio, unless $P = NP$. Our $O(\log n)$-approximation algorithm is also known to be the best possible unless NP has algorithms that run in "quasi polynomial time", meaning $2^{\text{poly}(\log n)}$ time. (Note: $2^{O(\log n)}$ time is another way of saying "polynomial time," so this is worse than polynomial time, but better than any 2^{n^ε} time bound.) We don't think NP has algorithms that are so fast, but with less conviction than $P \neq NP$.

Note also that even though our "exact" MAX 3-SAT problem

is NP-complete and has a $\frac{7}{8}$-approximation algorithm, it does <u>not</u> follow that every problem in NP has a $\frac{7}{8}$-approximation algorithm—cf., $O(\log n)$ is the best possible for Set Cover. This is because, if we look at the reduction that actually establishes 3-SAT to be NP-complete, the clauses don't naturally correspond to the objective of the original problem, so satisfying $\frac{7}{8}$ of them doesn't help us to solve the original problem in any clear way.