CS 124 Section #11 **Approximation Algorithms** 4/17/17

1 Summary of Approximation Algorithms from Lecture

1.1 k-Approximations

Approximation algorithms help us get around the lack of efficient algorithms for NP-hard problems by taking advantage of the fact that, for many practical applications, an answer that is "close-enough" will suffice. An approximation ratio, k, gives the ratio that bounds the solutions generated by our k-approximation.

For maximization problems, we want: $\text{APP} \geq \frac{1}{k} \cdot \text{OPT}$ and for minimization problems, we want $\text{APP} \leq k \cdot \text{OPT}$, with k being as close to 1 as possible.

1.2 Examples of approximation algorithms

- **Minimum Vertex Cover**: Given a graph $G = (V, E)$, can you find the minimal cardinality set of vertices $S \subseteq V$ such that for all $(u, v) \in E$, we have at least one of $u \in S$ or $v \in S$.

  A 2-approximation algorithm for this problem is relatively straight forward. Just greedily find a $(u, v) \in E$ for which $u \notin S_i$ and $v \notin S_i$, and construct $S_{i+1} = S_i \cup \{u, v\}$. We have $S_0 = \emptyset$. When not such (u, v) exist, return S.

  Furthermore, because this is a 2-approximation, it can be shown that $|S| \leq 2|\text{OPT}|$ where OPT is a minimum set cover.

- **Maximum Cut**: Given a graph $G = (V, E)$, can you create two sets S, T such that $S \cap T = \emptyset, S \cup T = V$, and the number of $(u, v) \in E$ that cross the cut is maximal?

  For this problem, we've covered two 2-approximation algorithms. The first is to randomly assign each $u \in V$ into either S or T. The second is to deterministically and iteratively improve our maximal cut by moving edges from S to T or vice-versa.

- **Euclidean Traveling Salesman Problem**: Give a set of points (x_i, y_i) in Euclidean space with l_2 norm, find the tour of minimum length that travels through all cities.

  A 2-approximation algorithm for this problem can be achieved by short-circuiting DFS on minimum spanning tree. Furthermore, the approximation can be improved to $\frac{3}{2}$-approximation. Both require the triangle inequality to hold.

- **Max Sat**: Find the truth assignment for variables which satisfies the largest number of OR clauses.

  We actually have two approximation algorithms. The first is by random coin-flipping (does best on long clauses). The second is randomized rounding after applying linear programming relaxation (does best on short clauses). We can also combine both for an even better solution.

2 Practice Problems

Exercise 1. Minimum Set Cover
We are given inputs to the minimum set cover problem. The inputs are a finite set $X = \{x_1, x_2, ..., x_n\}$, and a collection of subsets $\mathcal{S}$ of X such that $\bigcup_{S \in \mathcal{S}} S = X$. Find the subcollection $\mathcal{T} \subseteq \mathcal{S}$ such that the sets of $\mathcal{T}$ cover X, that is:

$$\bigcup_{T \in \mathcal{T}} T = X$$

Recall *minimum set cover* seeks to minimize the size of the selected subcollection (minimize $|\mathcal{T}|$).

(a) Formulate the set cover problem as an integer linear program.

(b) Let k be the number of times the most frequent item appears in our collection of subsets, $\mathcal{S}$. Use linear program relaxation to obtain a k-approximation algorithm for minimum set cover

Exercise 2. Maximum Set Packing

In the maximum set packing problem, we are given sets $S = \{S_1, S_2, \ldots, S_n\}$ each of which is a subset of a universe X. The goal is to find $T = \{S_{i_1}, S_{i_2}, \ldots, S_{i_t}\}$ such that all the S_{i_j}'s are mutually disjoint and $t = |T|$ is as large as possible.

(a) Show that maximum set packing is NP-complete by reducing from maximum independent set. In fact, show that there is a bi-directional reduction between these two problems.

(b) Suppose now that $|S_i| \leq k$ for all $i \in \{1, 2, \ldots, n\}$. A very simple greedy algorithm for this problem is to choose an arbitrary set $S_i \in S$, add it to T, remove all sets in S whose intersection with S_i is non-empty, and recurse until S is empty. Prove that this greedy algorithm achieves an approximation ratio of k.

(c) Consider a local search algorithm where we first run the greedy algorithm described in (b) to get an initial guess on T. Then, we keep on trying to increase the number of sets in T by removing one element of T and replacing it with two elements of $S - T$ such that the sets of T are still pairwise disjoint. Prove that this local search algorithm achieves a $\frac{k+1}{2}$ approximation ratio.

Exercise 3. Subset Sum

Suppose you are given a set of positive integers $A = \{a_1, a_2.....a_n\}$ and a positive integer b. A subset $S \subset A$ is called feasible if the sum of the numbers in S does not exceed b. You would like to select a feasible subset S of A whose total sum is as large as possible. You may assume that $a_i \leq b$ for all i.

(a) Show that the problem is NP-Hard.

(b) Consider the following greedy algorithm: Start with $S = \{\}$, which is certainly feasible. For $i = 1$ to $i = n$, if we can add a_i to S if doing so would make S still feasible. Otherwise, we do nothing with a_i. Show that this greedy algorithm can do arbitrarily poorly compared to the optimal solution. That is, show that for every $c > 0$, there exists a possible input (A, b) to this problem where the greedy algorithm achieves a sum that is c times smaller than the optimal sum.

(c) Show that if we first sort A in $O(n \log n)$ time, then the greedy algorithm from (b) achieves a 2-approximation. Give an example of an input (A, b) where this modified algorithm's solution is almost exactly a factor of 2 worse than the optimal solution.

(d) Modify the greedy algorithm from (b) in a small way to give an $O(n)$ 2-approximation to the subset sum problem. You will not be able to sort A.

(e) Describe a dynamic programming algorithm that solves this problem exactly and analyze its run-time.

Exercise 4. We know that that all of NP-complete reduce to each other. It would be nice if this meant that an approximation for one NP-hard problem would lead to another. But this is not the case. Consider the case of Minimum Vertex Cover, for which we have a 2-approximation. You can check quite easily that C is a cover in a graph $G = (V, E)$ if and only if $V - C$ is an independent set in V .

(a) Explain why this does not yield an approximation algorithm that is within a constant factor of optimal for Maximum Independent Set. That is, show that for any constant c, there exists a graph for which even if we obtain a 2-approximation of the Minimum Vertex Cover, the corresponding independent set is not within a factor of c of the Maximum Independent Set.

(b) The Maximum Independent Set problem and the Maximum Clique problem are related in the following way: an independent set of a graph G is a clique in the complement of G. (The complement of G is the graph that contains exactly the edges that are not in G.) Does an approximation algorithm (that is, an algorithm within a constant factor of the optimal) for the Maximum Clique problem yield an approximation algorithm (within a constant factor of optimal) for Maximum Independent Set?