CS264: Homework #3

Due by midnight on Wednesday, October 15, 2014

Instructions:

(1) Form a group of 1-3 students. You should turn in only one write-up for your entire group.

(2) Turn in your solutions at `http://rishig.scripts.mit.edu/cs264-bwca/submit-paper.html`. You can contact the TA (Rishi) at `bwca-staff@lists.stanford.edu`. Please type your solutions if possible and feel free to use the LaTeX template provided on the course home page.

(3) Students taking the course pass-fail should complete 5 of the Exercises and can skip the Problems. **Students taking the course for a letter grade should complete 8 of the Exercises. We'll grade the Problems out of a total of 40 points; if you earn more than 40 points on the Problems, the extra points will be treated as extra credit.**

(4) Write convincingly but not excessively. Exercise solutions rarely need to be more than 1-2 paragraphs. Problem solutions rarely need to be more than a half-page (per part), and can often be shorter.

(5) You may refer to your course notes, and to the textbooks and research papers listed on the course Web page *only*. You cannot refer to textbooks, handouts, or research papers that are not listed on the course home page. Cite any sources that you use, and make sure that all your words are your own.

(6) If you discuss solution approaches with anyone outside of your team, you must list their names on the front page of your write-up.

(7) Exercises are worth 5 points each. Problem parts are labeled with point values.

(8) No late assignments will be accepted.

Lecture 5 Exercises

Exercise 13

Prove that for the unweighted special case of the maximum-weight independent set (MWIS) problem (with $w_v = 1$ for all $v \in V$), the randomized greedy (RG) algorithm returns a solution with expected size at least $1/(\Delta_{avg} + 1)$ times the number n of nodes, where $\Delta_{avg} = 2m/n$ is the average degree of the graph.

[Hint: use the convexity of the function $f(x) = 1/x$ for $x > 0$.]

Exercise 14

Prove that our analysis of the recoverable value obtained by the RG algorithm is tight in the following sense. For every constant $c > 1$, there is a graph $G = (V, E)$ and weights $w_v \geq 0$ for $v \in V$ such that the expected weight of the solution returned by the RG algorithm is *not* at least

$$\max_I \sum_{v \in I} w_v \cdot \min\left\{1, \frac{c}{\deg(v) + 1}\right\},$$

where I ranges over all independent sets of G.

Exercise 15

This problem gives a derandomized variant of the RG algorithm from lecture. Specifically, consider the following deterministic greedy (DG) algorithm:

1. $S = \emptyset$.
2. While G is non-empty:
   (a) Let v be a vertex of G maximizing $w_v/(\deg(v)+1)$, where $\deg(v)$ denotes the degree of v in the current graph G.
   (b) Add v to S.
   (c) Delete v and all of its neighboring vertices from G.
3. Return S.

Prove that the DG algorithm is guaranteed to output an independent set of weight at least $\sum_{v \in V} \frac{w_v}{\deg(v)+1}$.

[Hint: there is a slick proof by induction, if you can figure out the right invariant.]

Exercise 16

Give a linear-time algorithm for computing the maximum-weight independent set of an acyclic graph (i.e., a forest).

[Hint: dynamic programming.]

Exercise 17

For the RG algorithm, we proved that the expected weight of the solution returned is at least

$$\sum_{v \in V} \frac{w_v}{\deg(v)+1}. \tag{1}$$

For the FR11 algorithm, we proved that the expected weight of the solution returned is at least

$$\sum_{v \in I} w_v \cdot \min\left\{1, \frac{2}{\deg(v)+1}\right\} \tag{2}$$

for every independent set I. Note that these two bounds are incomparable — the terms in (2) are bigger, but (1) sums over more terms.

Prove that the FR11 algorithm also satisfies the guarantee in (1).

Exercise 18

The point of this exercise is to give a parameterized analysis of the performance of a natural greedy algorithm for the Knapsack problem. Recall that in the Knapsack problem, you are given n items with nonnegative values $v_1, \ldots, v_n$ and sizes $s_1, \ldots, s_n$. There is also a knapsack capacity C. The goal is to compute a subset $S \subseteq \{1, 2, \ldots, n\}$ of items that fits in the knapsack (i.e., $\sum_{i \in S} s_i \leq C$) and, subject to this, has the maximum total value $\sum_{i \in S} v_i$.

Consider the following greedy algorithm:

1. Order items by "density," meaning reindex them so that $\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \geq \cdots \geq \frac{v_n}{s_n}$.
2. Return the largest prefix $\{1, 2, \ldots, j\}$ that fits in the knapsack (i.e., with $\sum_{i=1}^{j} s_i \leq C$).

Suppose that all items are *small* in the sense that, for a parameter $\alpha \in (0,1)$, $s_i \leq \alpha C$ for every i. Prove that the solution returned by the greedy algorithm has total value at least $(1-\alpha)$ times that of an optimal solution.

[Hint: as a thought experiment, what can you say about the greedy solution if it is allowed to pack items fractionally (where packing e.g. two-thirds of an item nets you two-thirds of its value)?]

Exercise 19

The point of this exercise is to give a parameterized analysis of the performance of a natural online algorithm for a scheduling problem. The set up is this: there are m machines, and you want to load-balance jobs on them in real time. Formally, a sequence of jobs arrives online (one-by-one), and when a job j arrives you must immediately assign it to one of the m machines, without knowing what the future jobs will be. Each job j has a *size* p_j. The *load* of a machine is the sum of the sizes of the jobs assigned to it. The *makespan* of a job assignment is the maximum load of a machine. We consider the objective of minimizing the makespan.

An obvious online algorithm is: when job j arrives, assign it to a machine that currently has the smallest load (breaking ties arbitrarily). Suppose that jobs are *small*, meaning that for every job j its size p_j is at most $\alpha \cdot (\sum_{i=1}^{n} p_i/m)$; recall m is the number of machines. Prove that the competitive ratio of the online algorithm is at most $1 + \alpha$.

[Hints: how does $(\sum_{i=1}^{m} p_i)/m$ compare to the smallest-possible makespan? How does it compare to the least loaded machine before the online algorithm schedules the last job?]

Lecture 6 Exercises

Exercise 20

We mentioned in lecture that it is NP-hard to approximate the k-median problem better than a factor of $1 + \frac{2}{e}$. This result is for worst-case instances, however, so we can't just assume without proof that it also applies to the special case of stable instances.

Prove that, without the additional large clusters restriction, for every choice of constants $c \geq 1$ and $\epsilon > 0$, it is NP-hard to approximate the k-median problem to better than a factor of $1+\frac{2}{e}$ in (c, ϵ)-stable instances.

[Hint: give a reduction from general k-median instances to stable ones. You might consider adding a bunch of "dummy" points, and modifying the parameter k accordingly.]

Exercise 21

Suppose we consider only the special case of (c, ϵ)-stable instances for which the large clusters assumption holds, where $c \geq 1$ and $\epsilon > 0$ are fixed constants, independent of the problem size. Show that the k-median problem can be solved exactly, in polynomial time, on such instances.

[Hint: since $\epsilon > 0$ is constant, your running time can depend on $1/\epsilon$, possibly in extreme ways. How many clusters can there be in such instances?]

Exercise 22

In lecture we assumed that the target clustering was also the optimal solution to the k-median instance. This exercise makes precise the idea that the optimal and target clusterings are more or less interchangeable in stable instances.

Formally, consider a k-median instance that is (c, ϵ)-stable with respect to the target clustering. Prove that the optimal and target clusterings agree on at least a $1 - \epsilon$ fraction of the points. Explain why this implies that the instance is also $(c, 2\epsilon)$-stable with respect to the optimal clustering.

Exercise 23

In lecture, in our proof of Claim 2, we implicitly assumed the following: if $\hat{C}_1, \ldots, \hat{C}_k$ is a k-clustering obtained from $C_1^*, \ldots, C_k^*$ by re-assigning ϵn points, then the two k-clusterings disagree on an ϵ fraction of the points. Show by counterexample that this is not true.

[Hint: recall that when measuring the agreement between two k-clusterings, we use the best-case correspondence between the two collections of clusters.]

Exercise 24

Continuing the previous exercise, show that if every cluster C_i^* of the initial clustering contains more than $2\epsilon n$ points, then the statement of the previous exercise holds.

[Remark: this is sufficient to complete the result from lecture, which was for instances that satisfy the large clusters assumption.]

Exercise 25

We described the algorithm from lecture as if we knew OPT, the optimal objective function value of the k-median instance (which is of course NP-hard to compute). Explain how to extend the algorithm so that OPT need not be known up front. (You should assume that the parameters ϵ and α are known.)

[Hints: try increasing "guesses" for OPT. How many distinct guesses do you need to try to cover all possibilities? How do you know when you've succeeded?]

Problems

Problem 6

(20 points) This problem is a parameterized running time analysis with respect to an input property for graph data. Specifically, this is a canonical example of how to solve problems that are NP-hard on worst-case graphs in polynomial time on graphs with bounded treewidth.

Recall that in the Steiner tree problem you are given an undirected graph with costs on edges, k vertices $t_1, \ldots, t_k$ are distinguished *terminals*, and the goal is to compute the minimum cost subgraph that spans all of the terminals (and possibly other vertices as well, if needed). This problem is NP-hard in general.

Give an algorithm that solves the Steiner tree problem correctly in every graph and, moreover, runs in time $O(f(t) \cdot g(n, k))$, where f is an arbitrary function, g is a polynomial function, and t denotes the *treewidth* of the input graph. You can use without proof any of the definitions, theorems, and algorithms from Sections 10.4 and 10.5 of the book *Algorithm Design*, by Kleinberg and Tardos.

Problem 7

(15 points) This problem will introduce you to *fixed-parameter tractability*, the idea that some NP-hard problems can be solved in polynomial time when the optimal solution is "small."

In the Hitting Set problem, there is a set $A = \{a_1, a_2, \ldots, a_n\}$ of elements and a collection $B_1, B_2, \ldots, B_m$ of subsets of A. A set $H \subseteq A$ is a *hitting set* for the collection $B_1, \ldots, B_m$ is H contains at least one element from each B_i — that is, if $H \cap B_i \neq \emptyset$ for every i. The natural decision version of this problem is NP-complete in general.

Suppose now that we are also promised that every set B_i has at most c elements, and we want to decide whether or not there exists a hitting set of size at most k. Give an algorithm that solves this problem with a running time of the form $O(f(c, k) \cdot p(n, m))$, where $p(\cdot)$ is a polynomial function, and $f(\cdot)$ is an arbitrary function that depends only on c and k, not on n or m. (Thus this algorithm is useful provided c and k are "small", even if n and m are big.)

Problem 8

The point of this problem is to introduce you to one popular way of parameterizing the "dimension" of geometric data. Let (X, d) denote a metric space (recall the definition from Lecture #6). A *ball* $B(x, r)$ with center $x \in X$ and radius $r \geq 0$ is the set $\{y \in X : d(x, y) \leq r\}$. The *doubling dimension* of a metric space is the smallest k such that, for every $x \in X$ and $r > 0$, the ball $B(x, r)$ is contained in the union of at most 2^k balls with radius $r/2$ (and centers of your choosing). Intuitively, in a space with low doubling dimension, there aren't too many "different directions of travel" from any fixed point.

(a) (4 points) In the *uniform* metric space, $d(x, y) = 1$ for every distinct pair x, y of points. Calculate the exact doubling dimension of the uniform metric on n points (as a function of n).

(b) (7 points) Formulate and prove a converse of sorts to (a): if a metric space (X, d) has doubling dimension at least k, it must be because there is a large subset $S \subseteq X$ such that the sub-metric (S, d) is "near-uniform."

(c) (5 points) Prove that doubling dimension generalizes Euclidean dimension, in the following sense. There are constants $c_1 > c_2 > 0$ such that, for all positive integers d, the doubling dimension of d-dimensional Euclidean space is between $c_2 d$ and $c_1 d$.

(d) (4 points) For a parameter $\delta > 0$, a *δ-net* of a metric space (X, d) is a subset $N \subseteq X$ of points such that (i) $d(x, y) \geq \delta$ for all distinct $x, y \in N$; and (ii) for every point $x \in X$, there exists $y \in N$ such that $d(x, y) < \delta$.

Suppose that N_1 and N_2 are δ- and $\delta/2$-nets of a metric space (X, d) with constant doubling dimension, with $N_1 \subseteq N_2$. Use (b) (or a variant of it) to argue that, for each $p \in N_1$, there is only a constant number (depending on the doubling dimension) of points $p' \in N_2$ within distance δ of p.

[Remark: To explain why this fact is useful in algorithmic applications, consider the task of computing a data structure for storing a point set such that subsequent nearest neighbor queries can be (approximately) answered quickly. One approach is to compute a nested sequence of δ_i-nets, where $\delta_i = \delta_{i-1}/2$. When a query point q arrives, its nearest neighbor p_1 in the first (smallest) net N_1 is computed by brute-force search; this exercise implies that, given p_1, there are not too many different candidates for q's approximate nearest neighbor in N_2; and so on. This allows the relevant data structures to have small space, and the query processing to spend a small amount of time searching each of the nets.]

Problem 9

The goal of this problem is to give a deterministic algorithm for the MWIS problem with a recoverable value of 2, matching the guarantee of the FR11 algorithm from Lecture #5. This algorithm is based on solving the following linear program:

$$\begin{array}{lll} \max & \sum_{v \in V} \frac{w_v x_v}{\deg(v)+1} & \\ \text{s.t.} & x_u + x_v \leq 1 & \text{for all } (u, v) \in E \\ & x_v \geq 0 & \text{for all } v \in V. \end{array}$$

You can assume without proof that an optimal solution $\mathbf{x}^*$ of this linear program can be computed in polynomial time.

(a) (2 points) Explain why you can assume, without loss of generality, that the input graph G has no isolated vertices.

(b) (2 points) Explain why the optimal objective function value of the linear program above is at least

$$\max_I \sum_{v \in I} \frac{w_v}{\deg(v) + 1},$$

where I ranges over all independent sets of the input graph G.

(c) (4 points) Prove that, without loss of generality, $\mathbf{x}^*$ is *half-integral*, meaning that $x_v^* \in \{0, \frac{1}{2}, 1\}$ for every $v \in V$.

[Hint: given an arbitrary feasible solution, show how to move it closer to a half-integral solution without degrading its objective function value.]

(d) (3 points) Let A and B denote the vertices with $x_v^* = 1$ and $x_v^* = \frac{1}{2}$, respectively. Let T denote the output of the algorithm of Exercise 15 on the graph $G[B]$. Let $S = A \cup T$. Prove that S is an independent set of G.

(e) (4 points) Using (a) and additional arguments, prove that $\sum_{v \in S} w_v$ is at least double the objective function value of $\mathbf{x}^*$. (Hence, by (b), this algorithm has recoverable value 2.)

Problem 10

Recall the FR11 algorithm from Lecture #5. An obvious generalization, for a parameter k, is the following. First, order the vertices $V = \{1, 2, \ldots, n\}$ randomly, as usual. For each $i = 1, 2, \ldots, n$, add i to a set T_k if and only if at most $(k-1)$ of i's neighbors precede it in the ordering. In Lecture #5, we saw that when $k = 1$ the set T_1 is an independent set and achieves recoverable value 1. We also saw that when $k = 2$, we can compute the MWIS S of the graph $G[T_2]$ induced by T_2, and this has recoverable value 2. The following problems investigate properties of the induced graph $G[T_k]$ when $k > 2$.

(a) (4 points) Recall that a k-coloring of a graph $G = (V, E)$ is an assignment of the vertices V to the colors $\{1, 2, \ldots, k\}$ such that the endpoints of every edge $e \in E$ are given different colors. Prove that for every $k \geq 1$, with probability 1, the induced graph $G[T_k]$ is k-colorable. Moreover, a k-coloring can be computed in polynomial time.

(b) (4 points) Prove that the expected weight of the maximum-weight independent set of $G[T_k]$ is at least

$$\max_I \sum_{v \in I} w_v \cdot \min\left\{1, \frac{k}{\deg(v) + 1}\right\},$$

where I ranges over all independent sets.

(c) (5 points) Building on the k-coloring computed in (a), give a polynomial-time algorithm that computes an independent set of $G[T_k]$ with total weight at least $2/k$ times that of the maximum-weight independent set.

[Hint: you can use without proof that the MWIS problem is solvable in polynomial time on bipartite graphs.]

(d) (2 points) Do (b) and (c) lead to an algorithm with a recoverable value better than 2?