15-451/651 Algorithm Design & Analysis

Spring 2023, Recitation #10

Objectives

- To practice coming up with and analyzing approximation algorithms.

Recitation Problems

1. **(Scheduling Jobs but with 1 Machine)** In lecture, we considered the job of minimizing the total makespan it takes to run n jobs on m machines, where the challenge was determining which jobs should be allocated to which machines.

   Here, we consider the variant of running n jobs on only a single machine, but where in addition to its processing time p_i and finish time T_i, each job has a release time r_i before which it cannot be worked on, and as our objective we want to minimize $\sum_i T_i$, the sum of their finish times. Intuitively, you can think of this objective as picking a job ordering to minimize the total time all clients spend waiting for their jobs to finish.

   Consider the following relaxation of this problem: a preemptive schedule is a schedule that allows us to stop a job in the middle of processing and resume the job from where we left off at a later time. We can compute the optimal preemptive schedule in polynomial time[1]. Let T'_i be the finish time of job i in the optimal preemptive schedule.

   (a) Scheduling with preemptive schedules is a relaxation of scheduling with non-preemptive schedules. In other words, any valid non-preemptive schedule is also a valid preemptive schedule. What does this imply about the relationship ($=$, $\leq$, or $\geq$) between $\sum_i T'_i$ and $OPT = \sum_i T^*_i$?

   (b) Let job j be the j^{th} job that the preemptive schedule finishes. Prove the following:

   $$T'_j \geq \max_{k=1}^{j} r_k \text{ and } T'_j \geq \sum_{k=1}^{j} p_k$$

[1] In fact, you can do it greedily. At any time, process the job with shortest remaining processing time. This is called the shortest remaining processing time (SRPT) rule. Try to show why this is optimal.

(c) Suppose that our non-preemptive schedule finishes jobs in the exact same order as our preemptive schedule. Show that $T_j \leq 2T'_j$. Conclude that this non-preemptive schedule is a 2-approx of OPT.

2. **(Randomized Rounding for Set Cover Approximation)** In class, we saw a 2-approximation of of VERTEX-COVER where we took our fractional solutions and rounded x_i up to 1 if $x_i \geq 1/2$ and down to 0 otherwise. However, this is not the only way to do LP relaxation and rounding, and in this problem we will explore other ways.

   (a) Write an LP relaxation for the SET-COVER problem:

   Given a universe $\mathcal{U} = \{1, 2, \ldots, n\}$ and a collection of sets $S_1, S_2, \ldots, S_m \subseteq \mathcal{U}$, pick sets $x_1, x_2, \ldots, x_k$ whose union is the entire universe

   $$\bigcup_{i=1}^{k} S_{x_i} = \mathcal{U}$$

   minimizing the number of sets picked k.

Our goal will be to use this LP to create a randomized approximation algorithm for SET-COVER. Suppose we solve the LP for optimal solution $\mathbf{x}^*$, and then interpret the fractional solution x_i^* as the probability of including set S_i in our cover.

(b) What is the expected size the cover our algorithm generates?

(c) For some element $u \in \mathcal{U}$, bound the probability that u is not covered.

Unfortunately, this results in a high probability of failure. To fix this, we'll pick random cover according to this method $\lceil \ln n \rceil$ times, and take the union of all these covers.

(d) Bound the expected size of this new cover generated from the union.

(e) For some element $u \in \mathcal{U}$, bound the probability that u is not covered.

(f) Now can you update this algorithm to be an approximation in expectation with no failure probability?

Further Review

1. **(Minimum Radius)** A city's electrical grid is represented by an undirected graph $G = (V, E)$. An edge represents two power stations which are directly connected to each other, and it takes one second to send power over an edge. Therefore, the minimum time it takes to send power from station u to station v is the graph distance between the two, or $d(u, v)$ seconds; this can require going through intermediate stations.

   The city wants to install $k \geq 1$ generators at some subset of the power stations H where $|H| = k$ (and k is fixed). Furthermore, they want to minimize the time it takes to send power from a generator to any other station in the city. In other words, noting that the maximum time it takes to send power to any station v is

   $$T_H(v) = \min_{h \in H} d(v, h)$$

   and the worst-case time for some generator configuration H is

   $$T(H) = \max_{v \in V} T_H(v)$$

   we are attempting to minimize $T(H)$ across all possible choices of H. However, computing this minimum is expensive. So, we settle for a 2-approximation in this problem.

   (a) Often, in coming up with approximation algorithms, it's a good starting point (or even correct) to start with a greedy algorithm.

   Start by placing a generator at some random vertex $v_1 \in V$. What might a greedy algorithm for this problem look like?

   (b) Show that your output set H satisfies $T(H) \leq d(a, b)$ for all $a, b \in H$ with $a \neq b$.

   Hint: Induction.

   (c) It turns out this algorithm does give a 2-approximation. Show this to be the case.

   Hint: Let t_{opt} be the optimal time across all H of size k, with H_{opt} being the corresponding choice of H. If we could get from any $h \in H_{opt}$ to H quickly, that would help, but we can't always do that. If we can't, then the pigeonhole principle helps.

2. **(SONET Ring Loading)** The following problem is a classical problem in telecommunications networks. We have a cycle with n vertices, numbered 0 through $n - 1$ clockwise around the cycle. We are also given a set of requests. Each request is a pair (i, j) where i is the source vertex and j the target vertex. The call can be routed either clockwise or counterclockwise through the cycle. The objective is to route the calls so as to minimize the load (total number of uses) of the most loaded edge of the cycle.

Write an linear program relaxation for the problem, and use it to give a 2-approximation algorithm using a rounding argument. Remember that a linear program relaxation is an LP such that if you could force some variables to be integers, you would solve the problem exactly.

3. **(Weighted Scheduling Jobs on 1 Machine)** Now consider the following weighted variant of the first problem: each job has an associated weight w_i and our objective is to minimize $\sum_i w_i T_i$.

   Unfortunately, it is NP-Hard to compute the optimal weighted preemptive schedule. Instead we'll tackle this problem via the following LP (where N is the set of jobs):

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} w_i T_i \\ \text{s.t.} \quad & T_i \geq r_i + p_i \qquad \forall j \in N \\ & \sum_{i \in S} p_i T_i \geq \frac{1}{2}(\sum_{i \in S} p_i)^2 \qquad \forall S \subseteq N \end{aligned}$$

   (Hint: The variables here are T_i's and everything else is a constant.)

   (a) Prove that this is a relaxation.

   (b) Given the optimal solution T^*, reorder the jobs s.t. $T_1^* \leq T_2^* \leq \ldots$. Show that if we set our schedule to finish jobs in this order, we have a 3-approx of OPT.

4. **(Planar Vertex Cover)** In class we saw a 2-approximation for the vertex cover problem by rounding its LP relaxation. We first solved the following LP with variables x_v for each $v \in V$, with the idea that, in an integer solution, $x_v = 1$ if v is in the vertex cover, and $x_v = 0$ otherwise.

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in V} w_v \\ \text{s.t.} \quad & w_u + w_v \geq 1 \qquad \forall \{u, v\} \in E \\ & w_v \geq 0 \qquad\qquad\quad \forall v \in V \end{aligned}$$

   When we solve the LP, we can get fractional values of x_v, so we *round* them by taking any vertices v for which $x_v \geq 1/2$. In this problem, we want to show that we can do better on planar graphs. First, we will need an interesting lemma about the solutions to this LP relaxation:

   (a) **(Optional)** Prove that for any vertex solution to the LP relaxation of vertex cover above, for any vertex v, the value of x_v is always in $\{0, \frac{1}{2}, 1\}$. This property is often called "half-integrality", because the vertex solutions to the LP are always either integral, or have variables equal to $\frac{1}{2}$, but nothing else.

Hints: Take a look at the optional material in the lecture notes for linear programming, where a very similar proof is given for the bipartite matching problem, which shows that its vertex solutions are always integral.

Now that we have this useful lemma, we are going to use another even more powerful result, arguably the most famous theorem in all of graph theory, the *four color theorem*, which says that *all planar graphs are four colorable*. Remember that a proper coloring of a graph is an assignment of a color to each vertex such that adjacent vertices are never the same color. A four coloring means that at most four different colors are needed. You may also the fact that such a coloring can be found in polynomial time.

(b) Combine this fact with the lemma from (a), and the fact that our favourite LP solving algorithms return a vertex solution, to produce a 1.5-approximation algorithm for vertex cover on planar graphs.

Hints: The standard rounding algorithm rounds up *every* vertex where $x_v = 1/2$, but this can be a little redundant if two adjacent vertices both have $x_v = 1/2$. Find a way to skip rounding some of the vertices, such that you are still guaranteed to have a valid vertex cover.