Design and Analysis of Algorithms **out:** April 16, 2015

Problem Set 8

Lecturer: Daniel A. Spielman **due: 2:30** PM, April 23, 2015

Instructions. Read Carefully

1. Make sure that your yale **netid** appears on **every** page of your problem set, and that it is written clearly.
2. The problem sets are due at the beginning of class. Solution sets will be provided in class. For this reason, late problem sets are not accepted.
3. You will **hand in each problem separately**, so make sure that they are not stapled together and that no problem appears on the back of another. There will be 4 boxes in the front of the class, one for each problem. Put each problem in the correct box.
4. Alternatively, you may submit the problem set electronically via the Classes server. If you do so, submit it as a **pdf** and make sure that your **netid** appears on **every** page. **Each problem must be in its own document.**
5. Cite any resources that you use, other than the textbook, TA and instructor.
6. You **must not** search the web for solutions to similar problems given out in other classes.

Collaboration Policy

You must write the solutions to the problems independently and in your own words.

However, you are allowed to discuss the problem sets in small groups of no more than 4 students. To ensure that you understand the solutions you submit, you are forbidden from taking written notes during your discussions. You must list at the top of the problem set everyone (other than course staff) with whom you have discussed the problem set. Failure to list people with whom you have discussed a problem set is considered a violation of academic honesty.

Problem 1: 3-Coloring as optimization

In this problem and the next, we will turn the problem of 3-coloring a graph into an optimization problem. The input will be a weighted graph $G = (V, E, w)$, where w assigns a positive weight to every edge. We will view a 3-coloring of V as a mapping c from V to $\{1, 2, 3\}$. We define the cost of c to be the total weight of all edges whose endpoints have the same color. That is,

$$\mathsf{cost}(c) = \sum_{(u,v)\in E : c(u)=c(v)} w(u, v).$$

The objective of our 3-coloring problem is to find a c of minimum cost. This is, of course, NP-hard. Your job is to find a polynomial-time algorithm that always finds a 3-coloring whose cost is at most

$$\frac{1}{3} \sum_{(u,v)\in E} w(u, v).$$

That is, a 3-coloring whose cost is at most one third of the total weight of all edges.

Prove that you algorithm satisfies this guarantee. If it is not obvious that your algorithm runs in polynomial time, explain why it does.

I suggest using a greedy algorithm that sets the colors of the vertices one-by-one.

Problem 2: Randomized 3-Coloring

In this problem, you must design *a different* algorithm for approximate 3-coloring. It must be randomized, like that used for maximum 3-SAT in the lecture from April 16.

Your algorithm find a coloring c so that

$$\mathsf{cost}(c) \leq \frac{2}{5} \sum_{(u,v)\in E} w(u, v)$$

with probability at least 1/2. It should run in linear time. State your algorithm, and prove that it satisfies this condition.

Note: I am asking for a weaker bound, 2/5 rather than 1/3, because I want a relatively high probability of satisfying this bound. While the algorithm that you found in Problem 1 also probably ran in linear time, the algorithm you find in this problem has other advantages. It should be much easier to parallelize. While we have not learned about parallel algorithms in this class, it is an important concept and this is my one small attempt at teaching you something that can be used to design them.

Hint: This is not a hard problem.

Problem 3: Makespan with Machines of Different Speeds

In this problem, we consider the makespan problem with machines of different speeds. The input will consists of n jobs, where job i requires t_i operations to complete, and k machines, where machine j can perform s_j operations per second. So, if job i is assigned to machine j, it will take time t_i/s_j. The Makespan problem is that of assigning jobs to machines so as to minimize the maximum completion time. That is, to partition $\{1, \ldots, n\}$ into k sets $A_1, \ldots, A_k$ so as to minimize

$$\max_j \frac{1}{s_j} \sum_{i \in A_j} t_i.$$

Give a polynomial-time algorithm that provides a 2-approximation of the minimum makespan, and prove that your algorithm satisfies this guarantee.

Hint 1: You can use a natural generalization of the algorithm we learned in class. But, it differs in an important way when there are machines of different speeds. Rather than adding a job to a machine that is doing as little work as possible, you should greedily minimize the completion time *after* a job is added to a machine. These two approaches are the same when all machines have the same speed.

Hint 2: I find the following inequality very useful. If $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ are positive numbers, then

$$\min_i \frac{a_i}{b_i} \leq \frac{\sum_i a_i}{\sum_i b_i}.$$

You may use this inequality, and trust me that it is true.

Hint 3: The analysis for this problem can be tricky. You can use hint 1 to generate an inequality. Also, note that some machines might be too slow to be used in the solution.

Problem 4: Integer Linear Programming

Write an integer linear program for Max-3-Sat: the problem in which one is given a set of clauses, each of which involves at most 3 terms, and must find the truth assignment that satisfies the maximum number of clauses.

Some of you may wonder why I ask you to do this, given that we cannot solve ILPs in polynomial time. The first answer is that we can often get good solutions to ILPs by solving the analogous LPs. The second is that there are packages like CPLEX and Gurobi that are pretty good at solving ILPs. So, this is a good way to solve many problems.

Hint: You should think of writing an ILP as like constructing a reduction between NP-hard problems. I suggest that you introduce a variable for every clause, in addition to the obvious variables upon which the clauses depend.