CS 182b
Spring 2011
Homework 5
Due Wednesday, February 23

This is the last assignment for this course! The next assignment will be the takehome exam (given out on Wednesday, February 23 and due back on Wednesday, March 2.)

1. **[25 Points] Traveling Salesperson!** In class, we saw an approximation algorithm for the Traveling Salesman Problem with Triangle Inequality (TSPTI) (aka "metric TSP"). This algorithm ran in polynomial time and guaranteed solutions that were within a factor of 2 of optimal. In this problem, you will design and analyze an approximation algorithm that finds solutions that are guaranteed to be within a factor of 1.5 of optimal! This is the best approximation ratio known for this problem at present.

   Consider an instance of TSPTI: an undirected completely connected weighted graph G where the edge weights satisfy the triangle inequality. A *minimum cost maximum matching* is a maximum matching such that the sum of the edge weights in that matching is minimized. There exists a polynomial-time algorithm that finds minimum cost maximum matchings in weighted undirected graphs. (The algorithm is a bit more complicated than the blossom algorithm that we've grown to know-and-love and you may assume that it exists.)

   (a) Recall that determining if an undirected graph contains a Hamiltonian cycle is NP-complete. However, determining if a graph contains an Eulerian cycle is polynomial-time solvable. Recall that an Eulerian cycle is a cycle (it starts and ends at the same vertex) that traverses each *edge* exactly once. A graph has an Eulerian cycle iff it is connected and every vertex has even degree. Based on this result, it's clear that determining if a graph is Eulerian can be done in polynomial time. Briefly explain how an actual Eulerian cycle can be found in such a graph in polynomial time.

   (b) Our new approximation algorithm first finds a minimum spanning tree (MST) in polynomial time. Then, it identifies the vertices that have odd degree in that MST. There are an even number of such vertices. So, we find a minimum cost matching in the subgraph of the original graph induced by these odd degree vertices. We add those edges to our MST, resulting in a connected graph in which every vertex has even degree. Complete

the description of the algorithm and prove that it gives a solution that is within 1.5 of optimal.

(c) Show that there exist arbitrarily large (with respect to the number of vertices) pathological TSPTI instances that force this algorithm to an approximation ratio that is arbitrarily close to 1.5. That is, show that the 1.5 ratio is asymptotically "tight".

2. **[25 Points] Finishing Up the Bin Packing Approximation Scheme!** In class we examined a polynomial time approximation scheme for Bin Packing. We showed that if all elements have size at least ϵ then in polynomial time we can find solutions which are at most $1 + \epsilon$ times optimal.

   Now we turn to those elements of size less than ϵ. We try to place them in the existing bins using the First Fit heuristic. If they all fit in the existing bins, we're done.

   The remaining case is that there are some items of size less than ϵ and they don't all fit in existing bins when using the First Fit heuristic. In this case, the First Fit heuristic was required to start some new bins. Your job is to show that in this case, the number of bins used by the algorithm is at most $(1 + 2\epsilon)OPT + 1$, assuming $\epsilon \leq 1/2$.

   Our analysis here will be entirely from scratch!

   (a) Let M be the number of bins used by this algorithm. Observe that with the exception of the last bin, all other bins are full to the extent $1 - \epsilon$ or more. Explain briefly why this is so.

   (b) Observe that the sum of the items sizes is at least $(M - 1)(1 - \epsilon)$. Briefly explain why.

   (c) Show the mathematical fact that if $\epsilon \leq \frac{1}{2}$ then

   $$\frac{1}{1 - \epsilon} \leq (1 + 2\epsilon)$$

   (d) Explain why the algorithm uses a number of bins at most $(1+2\epsilon)OPT+1$.

   (e) Show that the complete algorithm still has running time that qualifies it as a PTAS.

3. **[25 Points] Taking 3SAT to the Max!** MAX 3SAT takes an instance of 3SAT (a boolean expression in 3-CNF form) and asks for the valuation which maximizes the number of satisfied clauses.

Here's the approximation algorithm for MAX 3SAT that we saw in class: Choose a literal (a "literal" is a variable or the negation of a variable) which appears the most number of times in the given expression (breaking ties arbitrarily). Make that literal true, thereby satisfying some clauses. Remove all of these satisfied clauses. In the remaining clauses, every occurrence of the negation of the literal is crossed-off, thereby reducing the number of "live" literals in some clauses. Repeat until there are no clauses with "live" literals remaining. Notice that in this way, the number of "live" literals in each clause starts off at 3 but can eventually drop to 0, at which point the clause is "dead" and thus unsatisfied!

(a) First, generalize this algorithm for an approximation algorithm for MAX k-SAT, the variant in which each clause contains exactly k literals. Your approximation ratio will depend on k and should approach 1 as k approaches infinity!

(b) You may be saying to yourself: "That's very cool, but we should be able to do even better!" You're right! We can do MUCH better. Consider the following algorithm for the MAX k-SAT problem: Give each clause of the C clauses a weight 2^{-k}. Now, repeatedly choose some variable which has not been assigned a value yet. Let x be this variable. Let C_x be the set of clauses containing x and let $C_{\overline{x}}$ be the set of clauses containing $\overline{x}$. (These sets are disjoint, since otherwise the clauses in the intersection are trivially satisfiable.) If the total weight of the clauses in C_x is at least as large as the total weight of the clauses in $C_{\overline{x}}$, then make x true. Otherwise make x false. In the former case, remove all clauses containing x and, for each clause containing $\overline{x}$, remove $\overline{x}$ from the clause and double that clause's weight. Do the analogous thing if x is made false. Prove that this algorithm is a polynomial-time approximation algorithm with approximation ratio of $\frac{2^k-1}{2^k}$. YOWZA! *Hint:* Keep track of the total weight of all clauses. What happens to the total weight after reach round of the algorithm? What is the weight of a clause when it "dies" ?

4. **[25 Points] Precedence Constrained Scheduling.** In class we say a $\frac{3}{2}$-approximation algorithm for the problem of task scheduling with precedence constraints on 2 processors. Describe a generalization to the algorithm that works for any positive integer m of processors greater than or equal to 2 and show that the approximation ratio of your algorithm is $2 - \frac{1}{m}$.