Algorithms
Computer Science 140 & Mathematics 168
Spring 2009
Homework 10b
Due Tuesday, April 7

1. **[20 Points] Bin Packing Revisited!** Recall that on the last assignment you proved that the Bin Packing Problem is NP-complete. In particular, the Bin Packing decision problem is formulated as follows: We are given a collection of objects $S = \{a_1, \ldots, a_n\}$, a function $s : S \to \mathbb{Z}_+$ indicating the "size" of each object, a bin capacity C such that C is greater than or equal to the size of the largest object, and a target t. The question is whether or not it is possible to pack all of the objects in S into t or fewer bins, each of capacity C.

   The corresponding optimization problem gets rid of the target t and asks us to pack the objects into the least number of bins. In this problem, we will consider an approximation algorithm called "First Fit" which works as follows: Initially, allocate just a single bin and number it 1. Consider the objects one-by-one in no particular special order (just whatever order the items are given to us). Place the object in the lowest number bin in which it fits. If the object doesn't fit in any of the bins allocated so far, allocate a new bin and give it a counting number that is the next available number.

   So, initially there is just one bin. We place items in that bin until we get to an item that doesn't fit in the bin's remaining capacity. We now allocate a new empty bin 2 and place our item there. Now, for each subsequent item, if it fits in bin 1 put it there. If not, try bin 2. If it doesn't fit there either, allocate a new bin numbered 3 and place the object there, etc.

   (a) Give a small example that shows that the First Fit algorithm can use more than the optimal number of bins.

   (b) Prove that First Fit uses at most twice the optimal number of bins. In other words, show that First Fit is an approximation algorithm with ratio at most 2. Your proof should be precise and rigorous.

   (c) Explain briefly why First Fit runs in polynomial time. Thus, you can conclude that First Fit is a polynomial time 2-approximation algorithm.

2. **[20 Points] Linear Programming Relaxation!** Recall that we described a 2-approximation algorithm for the Vertex Cover problem - that is, an algorithm

that finds solutions that are at most 2 times as large as optimal. In this problem, we'll use a different (and extremely versatile) approach to finding a 2-approximation algorithm for this problem.

Here's the idea. We first reduce the Vertex Cover optimization problem into an Integer Linear Programming optimization problem. We do so by introducing a variable x_i for each vertex v_i in the graph. We constrain x_i to be 0 or 1 in the ILP (that's easy!) to represent not including or including, respectively, the vertex v_i in the vertex cover. For every edge $\{v_i, v_j\}$ in the graph, we introduce the constraint $x_i + x_j \geq 1$ to indicate that at least one of the two corresponding vertices must be selected for the vertex cover. (Of course, we convert those constraints into standard form.) Finally, we wish to minimize the objective function $x_1 + \ldots + x_n$.

The problem, of course, is that we're now left with an ILP problem and ILP is NP-complete itself! Here's the amazing trick: We now use a polynomial-time Linear Programming algorithm (rather than Integer Linear Programming). Notice that the integrality constraint is now relaxed. Therefore, this technique is called "Linear Programming Relaxation". Unfortunately, the solution found by the linear programming algorithm will permit each x_i to be some arbitrary rational number between 0 and 1. We now round each x_i in the linear programming solution to its nearest integer value: If $x_i < 0.5$ we round it down to 0 and if $x_i \geq 0.5$ we round it up to 1. (Notice the slight asymmetry in the rounding algorithm - that's important!).

Professor T. Ruth Teller (Professor I. Lai's nemesis) claims that the resulting solution corresponds to a valid vertex cover and, moreover, that vertex cover is at most twice as large as the optimal vertex cover. Your job is to prove this.

3. **[20 Points] Tightness of the TSPTI Approximation Algorithm!** In class we showed a polynomial time approximation algorithm for the Traveling Salesman Problem with Triangle Inequality (TSPTI). Recall that this algorithm first builds a minimum spanning tree. Then, it starts at some arbitrary vertex and "walks around the perimeter" of the minimum spanning tree. It then takes "shortcuts" when necessary in order to ensure that the tour visits each vertex exactly once. The ratio bound for this algorithm was also shown to be at most 2. Go back through your notes and make sure that you understand why this algorithm finds a solution that is no more than 2 times optimal.

   Professor Lai believes that the ratio of 2 is very sloppy and that we should be able to make a better guarantee for the performance of this algorithm. In this

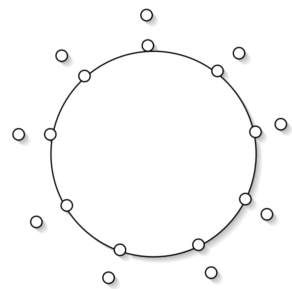

Figure 1: Points on the Euclidean plane. The inner set of points are on the perimeter of a circle of radius 1. The outer set of points are on the perimeter of a circle of radius $1 + \epsilon$.

problem you will show that, as usual, Professor Lai is wrong. In particular, show that for any $\delta > 0$ it is possible to construct an instance of TSPTI such that the approximation algorithm finds a solution that is *at least* $2 - \delta$ times larger than optimal.

Hint: Construct an instance of TSPTI where the points are arranged as shown in Figure 1 and use the Euclidean distance metric. In this graph, there are n points on the perimeter of a circle of radius 1 and n points on the perimeter of a circle of radius $1 + \epsilon$. Your goal is to show that for any $\delta > 0$ there exist appropriate values of n and ϵ such that when our approximation algorithm is used, the solution found will be at least $2 - \delta$ worse than optimal. You don't need to explicitly find ϵ for any δ, but just explain why such an ϵ must exist. In other words, don't do all the trigonometry to establish the value of ϵ as a function of δ.

It's critical that you understand that our approximation algorithm has some inherent "arbitrariness": Once a minimum spanning tree is computed, it chooses an arbitrary vertex in the graph. Next, it chooses one of the two directions arbitrarily and starts circumnavigating the spanning tree in this direction, taking shortcuts when vertices are visited twice. Our analysis shows that the resulting traveling salesman tour is no worse than twice optimal, regardless of the choice of vertex or orientation made by the algorithm.

4. **[20 Points] The Disk Storage Problem!** First the gratuitious story. You have been hired by Moon Microsystems to work on their new operating system, Lunaris. When Lunaris does a backup, it typically uses two large backup disks

and a tape drive. Since disks have lower access time than tape, it is desirable to store as many files as possible on disk and the remainder will go on tape. Let $F = \{f_1, \ldots, f_n\}$ denote the n files to be backed up and let ℓ_i denote the length of file i. Without loss of generality, let $\ell_1 \leq \ell_2 \leq \ldots \leq \ell_n$.

There are two identical disks, each with storage capacity of L. A file stored on disk cannot be broken up - it must be stored entirely on one of the two disks. The Disk Storage Optimization Problem is to store the maximum number of files from F onto the disks.

Your boss has suggested that you implement a greedy algorithm for the Disk Storage Optimization Problem: Since the files in F appear in non-decreasing order of size, simply go through F from shortest file to longest file, putting as many of the files on the first disk as possible. When the first disk fills up, continue iterating through F putting as many of the remaining files as possible on the second disk.

(a) Give an example that demonstrates that the greedy algorithm does not necessarily find an optimal solution. That is, give a set of specific file sizes, show the solution found by the greedy algorithm, and then the optimal solution. Show that the solution found by the greedy algorithm is not optimal.

(b) State the decision problem corresponding to this optimization problem. Then prove that the decision problem is NP-complete. Use a reduction from an NP-complete problem that we have seen in class or on a previous homework assignment.

(c) Prove that the greedy algorithm is an approximation algorithm which finds solutions that are at most 1 smaller than optimal. This is really neat, since all of the approximation algorithms we've seen so far were some *multiplicative* factor worse than optimal (typically 2 times optimal). Here, the algorithm finds solutions which are an *additive* amount worse than optimal!

5. **[20 Points OPTIONAL Bonus Problem] An Even Better Approximation Algorithm for TSPTI!**

In class, we saw an approximation algorithm for the Traveling Salesman Problem with Triangle Equality (TSPTI). 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!

Here are the ingredients that you'll need:

- Consider an undirected weighted graph G. A *matching* in G is a subset of the edges, no two of which share a common vertex. A *maximum matching* is matching of maximum total size. 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 complicated, but you may assume that it exists.
- 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 from Math 55 that an Eulerian Cycle is a cycle (it starts and ends at the same vertex) that traverses each *edge* exactly once. Refer to your Math 55 notes to remind yourself of how we can determine, in polynomial-time, whether or not a graph has an Eulerian Cycle. (You proved this in Math 55, even if you didn't explicitly use the words "polynomial-time" in that proof.)

Using these ingredients, and any other results from class, describe an approximation algorithm for TSPTI that achieves a ratio bound of 1.5. **Then**, show that there exist 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".