Advanced Algorithms, Fall 2014

Prof. Bernard Moret

Homework Assignment #5: Solutions

Question 1.

Given a set of points $\{x_1, \ldots, x_n\}$ on the number line, we want to find the smallest set of unit-length closed intervals that contains all of the given points, but we will approach the problem only with the following greedy algorithm: choose the remaining interval that covers the maximum number of as-yet-uncovered points until all points are covered. Analyze the performance of this algorithm in terms of the quality of the solution returned (will this algorithm always find the optimal solution? if not, does it guarantee an approximation ratio and, if so, do you have the tightest ratio possible?); provide proofs and/or counterexamples as necessary.

The algorithm is not optimal; a trivial example is the set of values $\{1,2,3,4\}$: the optimal solution, of course, is to use the interval $[1,2]$ and the interval $[3,4]$, but the algorithm could instead start by choosing the interval $[2,3]$ and then be forced to use two more intervals to cover 1 and 4, as they are too far apart to be covered by a single interval. This tiny example can be repeated over and over, separating each copy by 2 units; thus, if there is an apprxoimation guarantee, it cannot be better than 1.5. We saw in a previous homework that the optimal solution sorts all points, then repeatedly places the next unit interval to start at the next uncovered point and removes the points covered by that interval. It is easy to verify that the greedy algorithm and the optimal one return the same solution on 3 or fewer points and that the example with four points given above is a worst-case for 4 points. (If an instance has two consecutive points separated by more than one unit, then it can be partitioned into two independent parts.) However, consider now the set of values $\{0.7, 1.0, 1.5, 2.0, 2.3, 2.6, 3.1, 3.6, 3.9, 4.2, 4.7, 5.2, 5.5\}$. The optimal algorithm covers this set of 13 points with 5 intervals, the first four each containing 3 points, the last containing a single point. It is not possible to get more than 3 points per unit interval, so a greedy algorithm can pick bad intervals with 3 points each, namely the 3 intervals with points $\{x, x+0.5, x+1\}$, then have to pick 4 more intervals, each with a single (new) point, for a total of 7. Increasing the construction (adding more 4-tuples of points of the form $\{y, y+0.5, y+1, y+1.3\}$, with y equal to the largest number of the previous set increased by 0.3, yields a system where a set of $4k+1$ points yields an optimal solution of $\lfloor \frac{4k+1}{3} \rfloor + 1$ intervals, whereas the greedy algorithm requires $2\lfloor \frac{4k+1}{3} \rfloor - 1$ intervals, for a ratio that is always less than 2, but reaches it asymptotically.

We claim that this is indeed the worst case, so that the greedy algorithm has an asymptotic ratio of 2. Note that we need only prove this for non-partitionable instances, where two consecutive points cannot be more than one unit apart. Note also that, if the optimal solution for such an instance uses k intervals, then the leftmost and rightmost points are at most $2k-1$ units apart and more than $k-1$ units apart. In order to reach a worst-case, the intervals initially picked by the greedy algorithm (which are, after all, covering as many points as those picked by the optimal algorithm) must "miss" some points and have to collect these points later at great cost; thus the worst case arises when each such point must be collected separately. Hence the structure of a worst-case example is one in which the intervals picked by the greedy algorithm come in two categories: intervals that have as many points as possible, but are separated by a short segment

with a single point in it, and intervals that cover these short segments, one at a time. That is exactly the structure described above. Note that the actual number of points in all but the last optimal intervals must equal the number of points in the first collection of greedy intervals (the disjoint ones), and must be at least 3, but is not otherwise constrained—the final ratio remains $\frac{2\alpha(k)-1}{\alpha(k)+1}$, although $\alpha(k)$ grows more slowly the more points we use per interval.

Question 2. Consider the following problem: you are given a number n and asked to produce n from 1 by a succession of arithmetic operations, using the fewest possible number of such operations. The two operations allowed are to increment the current value by 1 and to double the current value.

Verify that the obvious greedy strategy (double until you no longer can, then use incrementations to finish) is not optimal. Then describe a different greedy strategy that does return the optimal solution.

To see that the obvious greedy strategy fails, pick $n = 2^k - 1$ for some large enough k. Then we can double $k-1$ times to generate 2^{k-1} and now are left with $2^k - 1 - 2^{k-1} = 2^{k-1} - 1$ incrementations, for a total of $2^{k-1} + k - 1$ operations. But we could have generated this number much faster by alternating doublings and incrementations: first double, then increment, repeating $k-1$ times, for a total of just $2(k-1)$ operations. Thus the obvious greedy algorithm not only fails, it fails very badly!

But now consider reaching 1 from n through the shortest possible sequence of operations, where the two operations allowed are decrementation by 1 and halving (division by 2). Obviously, any sequence of operations optimal for this problem can be transformed in a sequence of operations optimal for the original problem: just reverse the sequence of operations, replace decrement with increment and halving by doubling. The greedy approach is to apply whichever operation is legal and results in the largest decrease; so, if the number is odd, we decrement it by 1—the only legal operation, but, if the number is even, we divide it by 2—as that always reduces the number (which is larger than 1) as much as decrementation. This greedy approach is optimal. (Note, among other things, that it yields the correct sequence for $2^k - 1$.) How can we prove it? The basic step is to prefer, under any circumstances, halving to decrementing. Suppose then we have a number of the form $4k+2$; if we halve it, following the greedy approach, we get $2k+1$ and must now decrement it to get $2k$, followed by one halving to get it to k, for a total of 3 operations. Conversely, if we choose decrementing over halving so we can get to a value that is easier to halve, we use two decrementations to get down to $4n$, and then two halvings to get down to n, for a total of 4 operations—more than with the greedy approach. If we considered $2^i \cdot k + 2$, the ratio would get better for larger values of i, but always remain larger than 1; and if we considered $2^i k + 2j$, with $2j < k$, then the ratio would only get worse with increasing j. Since it is obvious that, with a simple power of 2, the optimal strategy is to keep halving, we have covered all cases.

Question 3. Define a family of graphs by this property: there exists some constant k such that any such graph has a subset of k edges whose removal turns the graph into a tree. Devise a linear-time algorithm to compute a minimum spanning tree on graphs in this family.

We can use one of two basic strategies, one based on the knowledge that we can greedily eliminate the longest edges and the other based on our transformation proof for the correctness of MST algorithms. Both, in the end, produce the same basic routine. If we use an approach based on greedy elimination of the longest edges, we need to find cycles in the graph and remove

the longest edge in the cycle, across all possible cycles. This may seem like a tall order—and would be, in a general graph—, but the number of cycles in a graph that is essentially a tree with a few added edges is not large. Moreover, it suffices to find only the simple cycles (cycles that do not pass through the same vertex more than once), because a general cycle is just a union of simple cycles. In fact, there is not even any need for considering all simple cycles: it is enough to consider a cycle basis, i.e., a collection of cycles from which all simple cycles can be built through disjoint union (exclusive-or) operations. (If a longest edge to be removed is found in some simple cycle, that edge is part of at least one of the cycles in the cycle basis, because disjoint unions cannot create new edges.) In our case, the number of simple cycles is solely a function of k and is thus itself another constant, so that an approach based on simple cycles will work, but an approach based on a cycle basis is much simpler, thanks to a fundamental result in graph theory: given a graph, a cycle basis can be found by the simple expedient of computing a spanning tree for the graph: the collection of all cycles induced by adding one of the remaining edges to the spanning tree forms a cycle basis. Thus, in our case, we need only look at k cycles. We start by computing any spanning tree for the graph, taking linear time to do so (we are not interested in a minimum spanning tree and so can use a simple graph traversal). We then add each edge not in the tree (k of them in all) and look at the one induced cycle, finding the longest edge in it; we compare the k edges thus found and remove the longest—this is the longest replaceable edge in the current graph. We then repeat with a spanning tree in which we (may have) exchanged one edge with one of the k edges that was not in it, and with just $k-1$ choices left. The running time for one step (one exchanged) is linear and we take k steps, for some constant k, so the overall running time is linear.

If we use an approach based on our transformation theorem, we come to a very similar strategy, but without any of the considerations about simple cycles and cycle bases: we know that we can transform any spanning tree into another by considering edges that are part of one, but not of the other, adding one such edge, and removing some edge on the induced cycle. We also know that removing the longest edge on the cycle cannot increase the total length of the spanning tree. Thus we start by find some spanning tree, which, as observed, we can do in linear time. We then identify the k edges of the graph that are not in the spanning tree, which can again be done in linear time. Now, we add one of the k edges to the tree and remove the longest edge from the induced cycle—a process that takes linear time in the worst case (if the cycle is long and its vertices have high degree). We repeat this step k times—note that, thanks to our transformation proof for the MST algorithm, we know that we do not have to select the next edge among the k in any particular order and that we can remove the longest edge from the one induced cycle, an improvement over the previous approach, which mimics a take-one-out greedy approach by removing the longest replaceable edge in the entire graph. The overall running time is $O(k|V|) = O(|V|)$, as desired. In a curious way, we are using the characteristics of a problem that can be solved optimally by greedy methods to develop what is, in effect, an iterative improvement method—but note that, on general graphs, this method takes quadratic time, which is very inefficient for the MST problem.

Question 4.
Prove that a connected bipartite graph on $2n+1$ vertices ($n \in N$) does not have a unique maximum matching. Given a connected bipartite graph on $2n$ vertices, what is the maximum number of edges it can have such that it still has a unique maximum matching? Provide a tight example for your analysis.

The keyword is "connected." A connected graph on $2n+1$ vertices has at least $2n$ edges. Notice that the statement is false for $2n$ vertices: we can set up a bipartite graph with n vertices on each side and a single path of $2n-1$ edges, with edges $\{u_i, v_i\}$ forming the matching and edges $\{u_{i+1}, v_i\}$ providing the connectivity, and such a graph clearly has a unique matching.

Since the number of vertices is odd, any matching must leave at least one vertex uncovered. Since we have at least 3 vertices (for $n = 1$) and the graph is connected, any maximum matching on any of these graphs has size at least 1. Let x be an uncovered vertex. Since the graph is connected, x is connected to some other vertex; that vertex cannot be one not covered by the matching, since then we could add the edge between x and that vertex to the matching and get a larger matching, a contradiction. Thus x is connected to a matched vertex, say u matched to v; but then we can replace the $\{u,v\}$ edge by the $\{x,u\}$ edge and obtain a new maximum matching.

Now, what is the largest number of edges that a connected bipartite graph of $2n$ vertices can have and still have a unique maximum matching? It is at least $2n-1$, using the example we started with, but can we do better? We cannot create a cycle that has half of its edges in the maximum matching, since we could flip the status of every edge in the cycle and obtain a new, different maximum matching. Since every edge we add to a connected graph forms a new cycle, this is a strong constraint. In addition, as we have seen with the graph of $2n+1$ vertices, we cannot afford to leave vertices unmatched, as any unmatched vertex (because of the connectivity requirement) immediately offers alternate edges for the maximum matching. This last observation suggests that we should look at a graph with n vertices on each side and a perfect matching. Now, how do we connect it and how many more edges can we add without creating a dangerous cycle? Perhaps surprisingly, we can pack $\Theta(n^2)$ edges! Let the graph be given by V_1 and V_2, each with n vertices, and the following collection of edges:

- For each i, $1 \leq i \leq n$, we add the edge $\{u_i, v_i\}$; this is our perfect matching.
- For each i, $1 \leq i \leq n$, and for each j, $i < j \leq n$, we add the edge $\{u_i, v_j\}$.

This graph has $\sum_{i=1}^{n} i = \frac{1}{2}n \cdot (n+1)$ edges. We claim that it has a unique perfect matching and that no bipartite graph on $2n$ vertices with more edges does.

The graph has a perfect matching by construction. That it is unique is easily seen by induction. The graph has two vertices of degree 1: v_1 and u_n; these must then be matched to, respectively, u_1 and v_n. Removing these two pairs and all incident edges simply leaves us with the same construction on $2(n-2)$ vertices, and eventually we come either to an empty graph (only one choice, the empty set) or a graph consisting of a single pair of vertices (only one choice: the edge connecting these two vertices). Thus the matching is unique.

It is clear that adding any edge to this construction creates an alternate perfect matching, but we need to show that *any* bipartite graph on $2n$ vertices with at least $1 + \frac{1}{2}n \cdot (n+1)$ edges has more than one maximum matching. By our previous observations, that is immediately true of any such graph where the matching leaves at least one vertex unmatched, so we need concern ourselves only with perfect matchings on bipartite graphs with n vertices on each side. Because any vertex of degree 1 dictates its own pairing and because it must be matched (we have a perfect matching), it can be removed along with its mate and all edges incident on the pair. Moreover, this process cannot reduce the graph to empty: at most, it will remove two vertices and n edges (the one edge incident to the vertex of degree one and a maximum of $n-1$ other edges incident on its matched vertex), so that successive applications would remove at most $\sum_{i=1}^{n} i$ edges—but we have at least one more edge than that. Thus we can assume that every vertex has degree at least 2. Pick some edge $\{u,v\}$ of the perfect matching; by the above, the graph contains vertices

u' and v' such that $\{u,v'\}$ and $\{u',v\}$ are also edges. If $\{u',v'\}$ is a matched edge, we are done, as we can then replace the two edges $\{u,v\}$ and $\{u',v'\}$ by the two edges $\{u,v'\}$ and $\{u',v\}$; this is the base case in an inductive process. Otherwise, we follow the matched edges from u' and v' and repeat. Because every vertex has degree at least two and because the number of vertices is n on each side, we eventually close a cycle and that cycle (as in the small cycle of 4 vertices in the base case) has every other edge in the matching, so that flipping the status of every one of its edges provides another perfect matching.

Question 5.
You are given a network where all edge capacities are larger than 1 in which the maxflow has some value $F \geq 1$. If the capacity of every edge is increased by exactly 1 unit, the new maxflow is some value $F + \Delta_1$; if the capacity of every edge is decreased by exactly 1 unit, the new maxflow is some value $F - \Delta_2$. Prove that we must have $\Delta_1 \leq \Delta_2$.

The simplest approach is to use the maxflow-mincut duality, but to think in terms of the number of edges in a cut rather than in terms of the capacity of a cut. So let us denote by $k_{\min}$ and $k_{\max}$ the smallest and highest numbers of edges among all mincuts in the original network. (Naturally all of these mincuts have capacity F.) Then the increment in the flow is bounded by the cut with the smallest number of edges—mincuts with more edges will not remain mincuts in the new network—so that we can write $1 \leq \Delta_1 \leq k_{\min}$. Similarly, after decrementing the capacity of every edge, original mincuts with fewer than $k_{\max}$ edges will not remain mincuts, as those with exactly $k_{\max}$ edges will have a smaller capacity. Hence we can write $k_{\max} \leq \Delta_2 < |E|$. (Note that both systems of inequalities are just bounds: we do not know that a new mincut after incrementing all edge capacities is an old mincut with a minimum number of edges, nor do we know that a new mincut after decrementing all edge capacities is an old mincut with a maximum number of edges. We do know, however, that these old mincuts are valid cuts in the new network and so serve as bounds.) It follows that we have $\Delta_1 \leq \Delta_2$.