Advanced Algorithms, Fall 2012

Prof. Bernard Moret

Homework Assignment #6: Solutions

Question 1. (Matching)
Consider the following simple model for mobile phones. We have n base stations and n phones, all of which are specified as points in the plane. The phones are said to be fully connected *if each phone can be assigned to a distinct base station, and the straight-line distance between the assigned pair of phone and station is no more than a given constant c.*

Suppose the user of the first phone drives from its original point along a line for k units of distance. As this phone moves, we have to update the assignment (possible several times) to keep all phones fully connected. Give an $O(n^3)$ running time algorithm to decide whether it is possible to keep all phones fully connected at all times during the driving. (Hint: a straight line cuts a circle in at most two points.)

First consider the original static state. We can decide whether the phones can be fully connected by building a bipartite graph and calculating the maximum matching. The nodes on one side of the bipartite graph are the n base stations, while the nodes on the other side are the n phones; a pair (base station, phone) is linked if the distance between them is no more than c. Clearly, the phones can be fully connected if and only if this graph has a perfect matching.

When a phone moves, consider the intersection points of the line of driving with the n circles, which define the boundaries of the covering areas of the base stations. We have at most $2n$ intersection points, which can be calculated and sorted along the line in $O(n \log n)$ time. These points, plus the starting point and the ending point of the segment, divide the line into at most $2n+1$ smaller disjoint segments. In different segments, the phone will be covered by different base stations.

We could run the above routine for each smaller segment, but this will require more than cubic time, since we have $O(n)$ segments and the maximum matching algorithm cannot be done in $O(n^2)$. However, we do not need to run the routine every time: instead, we can maintain a perfect matching and update it when situation changes. We first build the bipartite graph G for the starting point, and run the maximum matching algorithm to check whether this graph has a perfect matching M. If yes, we store G and M; otherwise, the whole algorithm terminates and retu rn false. Then we process the intersection points along the segment one by one. When we meet a new point, only one base station changes. If a new base station becomes available, we just need to add a new edge to G w hile keep M unchanged; if a base station becomes unavailable, we first remove the corresponding edge e from G, and then check whether $e \in M$. If $e \notin M$, we do not need to update M; otherwise, we remove e from M, which will create two unmatched nodes, and then traverse the graph by breadth-first search to try to find an augmenting path. If such augmenting path is found, we obtain a new perfect matching; otherwise, we know that the current bipartite graph does not have a perfect matc hing, and in this case, we just terminate the algorithm and return false. The dominating operation in processing each intersection point is breadth-first search, which costs $O(|E|) = O(n^2)$. Since we have $O(n)$ intersection points, the algorithm runs in $O(n^3)$.

Question 2. (Stable Matching)
Prove that a stable matching is both man-optimal and woman-optimal if and only if it is the unique stable matching for the problem.

One direction is trivial: if the matching is unique, it is by default man-optimal and also woman-optimal.

Now let us assume that some stable matching M_1 is both man-optimal and woman-optimal, but that there exists a second, different stable matching M_2. Since the two matchings are different, M_2 cannot be man-optimal, nor can it be woman-optimal. Let m_i be the first man who is not engaged to the same woman in both M_1 and M_2 and denote his mate in M_1 by w_1 and in M_2 by w_2. Since M_1 is man-optimal, m_i prefers w_1 to w_2—in fact, he prefers w_1 to any other woman he can marry in a stable matching. Similarly, since M_1 is also woman-optimal, w_1 prefers m_i to any other man she can marry in a stable matching. But then M_i and w_1 cause an instability in M_2: m_i prefers w_1 to w_2 (to whom he is married) and w_1 prefers m_i to any man to whom she could be married in a stable matching. Thus M_2 cannot exist and M_1 is the unique stable matching.

Question 3. (Stable Matching)
Consider the stable matching problem in the case where ties are allowed in the preference lists.

1. *A* strong instability *in a perfect matching S consists of a man m and a woman w such that m and w prefer each other to their partners in S. Does there always exist a perfect matching with no strong instability?*

   Give a polynomial-time algorithm that is guaranteed to find a perfect matching without strong instability and prove the correctness of your algorithm, or give an instance for which every perfect matching has a strong instability.
2. *A* weak instability *in a perfect matching S consists of a man m and a woman w such that m prefers w to his partner in S and w either prefers m to his partner in S or likes the two men equally; or w prefers m to her partner in S and m either prefers w to her partner in S or is likes them equally. Does there always exist a perfect matching with no weak instability?*

   Give a polynomial-time algorithm that is guaranteed to find a perfect matching without weak instability, or give an instance for which every perfect matching has a weak instability.

1. The proposal algorithm we described will work fine; we simply add that, in case a woman currently engaged receives a proposal from an equally ranked man, that man is rejected.
2. In this case, we can produce a counterexample, showing that a perfect matching without weak instability does not always exist. Consider an instance with two men, m_1 and m_2, and two woman, w_1 and w_2. Both m_1 and m_2 prefer w_1 to w_2, and w_1 and w_2 have both listed m_1 and m_2 as indifferent. Without loss of generality, assume that m_1 is engaged with w_1 and m_2 is engaged with w_2. Now m_2 and w_1 form a weak instability, because m_2 prefers w_1 to his current partner w_2, and w_1 is indifferent to m_1 and m_2.

Question 4. (Network Flow)
We are given a directed network $G = (V,E)$ with a single specified good *node $g \in V$ and a set of* bad *nodes $B \subset V$ (naturally, we have $g \notin B$). We want to disconnect bad nodes from g by removing edges, but face a tradeoff: we want to remove as few edges as possible and yet we want to remove as many bad nodes as possible.*

In consequence, we want to maximize the objective function $|f(S)| - \alpha \cdot |S|$, where S is the subset of E to remove, $f(S)$ is the set of bad nodes that cannot be reached from g in the subgraph $(V, E-S)$, and α is a positive constant. Give a polynomial-time algorithm to find a subset $S \subset E$ to maximize this objective function, and prove the correctness of your algorithm.

We can reduce this problem to a standard max-flow problem. First, we rewrite the objective function $|f(S)| - \alpha \cdot |S|$. Recall that $f(S)$ is defined as the set of bad nodes that cannot be reached fro m g in subgraph $(V, E-S)$. Thus, we can write $|f(S)| = |B| - |h(S)|$, where $h(S)$ is the set of bad nodes that can be reached from g in subgraph $(V, E-S)$. Since $|B|$ is a constant, we only need to maximize $-|h(S)| - \alpha \cdot |S|$, or equivalently, minimize $|h(S)| + \alpha \cdot |S|$.

We build the network based on G as follows. Choose g as source node add a new sink node t to G. Link all bad nodes to t and set the capacity of each such edge to 1. The capacity of all edges in E are set to α.

Now we show that finding the minimum *g-t* cut of the network is equivalent to finding a subset S to minimize $h(S) + \alpha \cdot |S|$. On the one hand, suppose S' is the minimum *g-t* cut of the network, and assume that S' is a combination of some edges in E, denoted as E', and some new added edges, denoted as N'. Thus, the value of this cut is $|N'| + \alpha \cdot |E'|$. Now we claim that $|N'| = |h(E')|$. In fact, if we remove the edges in E', those bad nodes covered by N' must be reachable from g—otherwise, we can remove those new edges in N' co rresponding to the bad nodes that cannot be reached from g to obtain a better cut, contradicting that S' is the minimum cut; besides, those bad nodes that are not covered by N' cannot be reached from g—otherwise, there will be a path from g to t that does not use any edge in S', contrad icting that S' is a cut. According to the definition of $h(\cdot)$, we have $|N'| = |h(E')|$. Thus, the value of S' is $\alpha \cdot |E'| + |h(E')|$, which gives a upper bound of the minimum value of the objective function . On the other hand, suppose S is the subset that minimize the objective functio n. We can add $|h(S)|$ new edges that corresponding to the bad nodes that can be re ached from g to S to make it a cut of the network. The value of this cut is $|h(S)| + \alpha \cdot |S|$, which also gives a upper bound of value of the minimum cut.
Thus, we can solve this problem by computing the minimum cut S' of the network—the value of S' is the minimum value of the objective function and $S' \cap E$ is the corresponding subset.