6.854 Advanced Algorithms

Problem Set 8

John Wang
Collaborators:

Problem 1-a: Suppose the optimum diameter d is known. Devise a greedy 2-approximation algorithm (an algorithm which gets k clusters each of diameter at most $2d$).

Solution: We start and pick an arbitrary center point x_1. This will be center of cluster c_1. We will place all points within a distance of d from x_1 into cluster c_1. In other words, for all points y such that $d(x_1, y) \leq d$, we will set $y \in c_1$. Now, we will continue this procedure k times. At the end, we will have formed k clusters, each of diameter at most $2d$.

I will now show the correctness of this 2-approximation. First, we know that each point x will be inside a cluster c of diameter d. Therefore, a cluster of diameter $2d$ centered at x will completely contain cluster c as a subset. This follows because any point in cluster c must be the center of the larger cluster, which implies that in any orientation of the point, the smaller cluster will be contained in the larger cluster. This means that OPT's cluster c will be contained in the approximation algorithm's cluster. It is clear by the triangle inequality that no point in each cluster is more than a distance of $2d$ away from another point in the same cluster.

From this, we see that each one of OPT's clusters k_i will be wholely contained by the approximation algorithm's clusters c_i. Thus, each point will be accounted for once the approximation algorithm terminates. Therefore, we see that the approximation alogirthm will find k clusters each of diameter at most $2d$ which will cover the entire space of points. Moreover, since we know that the optimum diameter is d, we see that this is a 2-approximation. □

Problem 1-b: Consider an algorithm which (k times) chooses as a center the point at maximum distance from all previously chosen centers, then assigns each point to the nearest center. By relating this algorithm to the previous algorithm, show that you get a 2-approximation.

Solution: If the algorithm picks points which are at least d apart from each other, then each center will belong to a different cluster k_i in OPT. This follows because clusters in OPT have a diameter of d, which means that if two points x and y have a distance $d(x, y) > d$, then x and y cannot belong to the same cluster. Now, if all k centers are d distance apart from each other, then we have created k clusters where no point in any of the clusters is further than d from each of the centers. This means the maximum diameter of the clusters will be $2d$.

Thus, all that needs to be shown for a 2-approximation is that the algorithm will pick center points which are at least d apart from each other. Suppose this is not true. Then after the algorithm has found p centers that are d apart from each other, there will no longer be any points which are d apart from all the previously chosen centers. This, however, implies that all of the remaining points can be allocated into some of the clusters which already exist. This means that we do not need all k of the centers for OPT's algorithm, which means we can decrease the maximum diameter (in the worst case, we can set the diameter to the second larger diameter, and make a cluster for the single point which is left out). This shows that OPT does not provide the optimal diameter, which is a contradiction. Therefore, we algorithm must pick center poitns which are at least d apart from each other and so the algorithm is a 2-approximation. □

6.854 Advanced Algorithms

Problem Set 8

John Wang
Collaborators:

Problem 2-a: Argue that the greedy algorithm (repeatedly take any edge that does not conflict with previous choices) can be implemented in linear time and gives a 2-approximation to the maximum (number of edges) bipartite matching.

Solution: First, we shall argue that repeatedly taking any edge that does not conflict with previous choices can be used to create an approximation algorithm that runs in linear time. The nodes are ordered arbitrarily, but they must be ordered so that edges originating from the same node are placed together. This ordering can be done in $O(n+m)$ time by using a list of lists for each node u, where all edges (u, v) with the same starting node u are grouped together.

Next, we take the a single edge from each list of lists and include it in the bipartite matching, and remove the entire list from the overall list. Additionally, we keep a hash of all the nodes v for which an edge (u, v) is incident upon. If we select an edge to delete where either node u or v is already used, then we throw away that edge and continue.

In total, we will examine no more than m edges, so the total algorithm runs in $O(m+n)$, which is linear. Next, we note that this is a 2-approximation.

Let us say that the edge that the approximation algorithm choose (u_1, v_1) is not the same as the edge OPT chooses for v_1. Namely, suppose OPT choose (u_0, v_1). Then in the worst case, OPT can choose a different edge (u_1, v_2) starting from u_1. However, we see that every incorrect choice that the approximation algorithm makes only affects 2 choices that OPT makes. Thus, since the approximation algorithm is still creating a matching, OPT do at maximum twice as well as the approximation algorithm. This follows because for each edge (u_1, v_1) selected by the approximation algorithm, OPT can only differ by two edges, which means OPT can only improve by twice as much. Thus, this algorithm is a 2-approximation. □

Problem 2-b: Generalize to argue that when edges has positive weights, the greedy algorithm (consider edges in decreasing order of weight) can be implemented in $O(m \log n)$ time and is a 2-approximation algorithm for maximum (total) weight bipartite matching.

Solution: It is clear that the algorithm can be implemented in $O(m \log n)$ time. For each node u for which there is an outgoing edge (u, v), take the highest weight edge of all outgoing edges. Sort all of these highest weight edges and put them into the matching. This is equivalent to examining the edges in decreasing order by weight because each source node u can only provide a single edge in the matching, which means that edge must be the highest weight edge leaving u.

To show that this algorithm is a 2-approximation, we make a similar argument as above. We note that whenever the approximation algorithm chooses an edges (u_1, v_1), OPT could have chosen two different edges, one starting on u_1 and one ending at v_1. However, we know that both these two possible edges have weight less than the edge (u_1, v_1), or else they would have been selected. This means that OPT can possibly have twice as large of total weight per edge that is selected by the approximation algorithm. It holds therefore, that this algorithm is a 2-approximation. □

Problem 2-c: Show that the same holds for a general (non-bipartite) max-weight matching problem.

Solution: The algorithm remains the same. We sort the edges in each node and select the highest weight edge form each node. We shall show that this algorithm still works on a general graph to create a 2-approximation.

Let s be the weight of the first edge which is removed (thus, s is the maximum edge weight on the graph). Let s correspond to the weight on edge (u, v). Then we know that when the algorithm removes edge (u, v)

and all incident edges on u and v, then at most two edges from the optimal matching will be removed. This follows because there can be at most two edges in the optimal matching which are incident on u and v.

However, we know that both of these edges must have weight less than or equal to s. Thus, the optimal matching for all edges incident on nodes u and v has weight less than or equal to $2s$. Recursively performing the same procedure for the other $n-2$ nodes shows that the optimal matching will have weight less than or equal to twice times the weight of the approximation algorithm. This means that we have provided a 2-approximation. Note that this proof only works when edge weights are non-negative. $\square$

6.854 Advanced Algorithms

Problem Set 8

John Wang
Collaborators:

Problem 3-a: Suppose you compute all-pairs shortest paths in G, and create a complete graph G' on the termainsl T where each hedge cost is equal to the shortest path between its (terminal) endpoints in G. Relate the cost of the minimum spanning tree in this graph to the cost of the optimum steiner tree in G.

Solution: We see that the optimal Steiner tree in G has a cost which is greater than half of the cost of the minimum spanning tree in G'. First, let us assume we have found the optimal Steiner tree S. Then, from this Steiner tree, we can construct a minimum spanning tree in G'.

We perform a DFS and visit all of the nodes in S starting from the root of the Steiner tree. The order in which the terminals in T are visited will be the order we can use to construct a minimum spanning tree. Suppose we visit terminals $v_1, v_2, \ldots, v_n \in T$ in that order during our DFS of S. Then we see that we are always moving along a shortest path from v_i to v_{i+1} since Steiner trees guarantee that we have a minimum cost span across all vertices. If this were not the case, then we could find some shorter path between v_i and v_{i+1} and exchange that path in the Steiner tree, which would find a more optimal Steiner tree. Since we were assuming the optimality of our Steiner tree, this is a contradiction.

Since we are always taking shortest paths between v_i and v_{i+1}, we know that we have a minimum spanning tree in G' between $v_i \forall i \in [1, n]$. Thus, our path in S has led to a minimum spanning tree in G'. Now, we shall show that the DFS in S traverses a path with with a cost at most twice of that of S. This is because the DFS visits each edge at most twice (once up and once down the DFS search). This shows that the cost of the spanning tree in G is greater than half of the cost of the minimum spanning tree in G'. □

Problem 3-b: Give a 2-approximation for the Steiner tree problem.

Solution:
□