COMP 360 - Winter 2016 - Assignment 6

Due: 6pm April 13th.

General rules: In solving these questions you may consult books but you may not consult with each other. You should drop your solutions in the assignment drop-off box located in the Trottier Building.

---

1. (25 Points) We are given a set P of n points on the plane, and a positive integer k. We want to partition these points into k sets such that the largest distance between any two points which belong to the same part is minimized. Show that the following is a 2-factor approximation algorithm.

   - Set $S = \emptyset$.
   - For $i = 1, \ldots, k$ do
   - Select the farthest point from S and add it to S.
   - EndFor
   - Put each point $p \in P$ in the same part as the closest point in S to p.

   (The distance from a point to S is the distance of the point to the closest point in S.)

   Solution: If the number of points is at most k, then the above algorithm is obviously optimal. So we can assume that $|P| > k$. Then let $p_1, \ldots, p_k$ be the points in S when the algorithm terminates and let p_{k+1} be the farthest point from S, and let r be the distance of p_{k+1} from S. Then every point in P is in distance of at most r to at least one of the points in S, and in particular in the partition that we obtain from the algorithm the largest distance between any two points that belong to the same part is at most $2r$. Thus the output of the algorithm is at most $2r$.

   Moreover, because of the way that the algorithm chooces the points in S, we ahve that $p_1, \ldots, p_{k+1}$ are in pairwise distance at least r. Hence in any partition (including the optimal one) at least two of $p_1, \ldots, p_{k+1}$ will belong to the same part. Hence the diamater of at least one part will be $\geq r$. Hence the optimal solution is at least r.

2. (25 Points) Problem 10 of Chapter 11: Suppose you are given an $n \times n$ grid graph G. Associated with each node v is an integer weight $w(v) \geq 0$. You may assume that all the weights are distinct. Your goal is to choose an independent set S of nodes of the grid, so that the sum of the weights of the nodes in S is as large as possible. (The sum of the weights of the nodes in S will be called its total weight.) Consider the following greedy algorithm for this problem.

   - Start with $S := \emptyset$.
   - While some node remains in G:
     - Pick a node v of maximum weight.
     - Add v to S.
     - Delete v and its neighbors from G

- Endwhile.

Show that this algorithm returns an independent set of total weight at least $\frac{1}{4}$ times the maximum total weight of any independent set in the grid graph G.

Solution: Since for every node v picked we remove the neighbors, the algorithm will not output any connected nodes thus the algorithm gives an independent set. Suppose that we pick a node v at some point in the algorithm. Let $v_1, \ldots, v_4$ be its neighbours. Note that none of $v_1, \ldots, v_4$ have been picked at this point (otherwise v would have been deleted). Since v has the maximum weight among the remaining vertices, we have $\text{weight}(v) \geq \text{weight}(v_i)$ for $i = 1 \ldots 4$. So

$$4\text{weight}(v) \geq \sum_{i=1}^{4} \text{weight}(v_i).$$

If the optimal algorithm doesn't choose v and chooses a subset (or all four) of the neighbors instead, then it could be at most 4 times better.

3. (25 points) Consider the triangle elimination problem. We are given a graph $G = (V, E)$, and want to find the smallest possible set of vertices $U \subseteq V$ such that deleting these vertices removes all the triangles (i.e. cycles of length 3) from the graph. Prove that the following algorithm is a 3-factor approximation algorithm for this problem:

   - While there is still a triangle C left in G:
   - Delete all the three vertices of C from G
   - EndWhile
   - Output the set of the deleted vertices

   Solution: Let $C_1, \ldots, C_m$ be the triangles that are found by the algorithm. The algorithm deletes $3m$ vertices. On the other hand any solution must remove at least one point from each one of these vertex disjoint triangles. Consequently, the optimal solution is at least m.

4. (25 Points) Given a set P of n points on the plane, consider the problem of finding the smallest r such that there exist 10 circles of radius r such that together they contain all the points in P. Design a PTAS algorithm for this problem. In other words, given any fixed $\epsilon > 0$, design an algorithm whose running time is polynomial in n, and its output is at most $1 + \epsilon$ times the optimal output.

   Solution 1: If $n \leq 10$ we are done. Otherwise we first run the 2-factor approximation algorithm for the k-center problem (with $k = 10$). Let t be the output of that algorithm. Then we know that the optimal radius r^* satisfies $t/2 \leq r^* \leq t$.

   Next divide the part of the plane that contains the points into a gird whose cells are $\frac{t\epsilon}{4} \times \frac{t\epsilon}{4}$. In the optimal solution every center is in distance at most t from at least one point in P (if that is not the case, then since $r^* \leq t$, the corresponding circle will not contain any points). Now if we consider all the grid points that are in distance at most t from at least one point in P then in total we have $\frac{8}{\epsilon} \times \frac{8}{\epsilon} \times n$ such points. If we try all the possibilities of choosing 10 centers among them then we have at most $(\frac{8}{\epsilon} \times \frac{8}{\epsilon} \times n)^{10}$ choices. We pick the best one and output it.

   It remains to show that our output radius is not larger than $(1 + \epsilon)r^*$. Let $c_1, \ldots, c_{10}$ be the optimal centers. Let $c'_1, \ldots, c'_{10}$ be such that c'_i is one of the corners of the grid cell that contains c_i. Note that the distance between c_i and c'_i is at most $\sqrt{2}\frac{t\epsilon}{4} \leq \frac{t\epsilon}{2}$. At some point our algorithm has checked $c'_1, \ldots, c'_{10}$, and since $\text{dist}(c_i, c'_i) \leq \frac{t\epsilon}{2} \leq r^*\epsilon$, for that particular choice of centers it suffices to consider circles of radius at most $r^* + r^*\epsilon \leq (1 + \epsilon)r^*$. Hence the output of the algorithm is at most $(1 + \epsilon)r^*$.