15-451 Assignment 07

Karan Sikka
ksikka@cmu.edu
Collaborated with Dave Cummings and Sandeep Rao.
Recitation: A
November 6, 2014

1: A Densely-Knit Community

(a) (b) (c) (d)

2: Large + Dense = Difficult

3: A Well-Separated Problem

(a) The problem is in NP because there exists a poly-time verifier as follows:

Define the proof of the solution as a K-element subset of X. We compute distances between each pair of distinct points and check that that they are greater than or equal to Δ. If all pairs satisfy the condition, then we verify that this is a solution. Otherwise it is not.

The Well-Separated problem is in NP-Hard because the independent set decision problem which is NP-hard reduces to it. The reduction is as follows:

Independent set
Given a graph $G = (V, E)$ and integer k,
we want to output YES
if there exists a set of vertices of size k
such that no two of them are adjacent.

To craft our input to the Well-Separated oracle,
we construct a set X from the vertices of G,
let $K = k$,
let $\Delta = 1.25$,
for all $i \in V$ let $d(i, i) = 0$,
for all $i, j \in V : i \neq j \wedge (i, j) \in E$ let $d(i, j) = 1$
for all $i, j \in V : i \neq j \wedge (i, j) \notin E$ let $d(i, j) = 1.5$

We pass this input to the well separated problem, which will return YES
if there exists a set of elements of size K where all distances are greater than 1.25.

Observe these elements in X map directly to vertices in V which are not adjacent
due to the way we constructed the input to the Well-Separated problem.

Also note that the construction of d correctly obeys the triangle inequality, because
two of the shortest distances $(1 + 1)$ is still greater than the longest distance (1.5).

(b) We will present the algorithm, prove a condition about it's correctness, and show that it runs in poly time.

Algorithm:
Call one set with separation at least $\Delta^*/2$ set C.

We will maintain a vector of all points which are potentially in C, initially containing all points.

We will also maintain a vector of points which we know are in C.

1. Pick a point u potentially in C that's not already in C, and examine how far away all other points are from it.
2. Remove all potential points not at least $\Delta^*/2$ from u from the set of points potentially in C.
3. Add u to the set of known points C.

Repeat for a total of K iterations,
resulting in a set C with K points at least $\Delta^*/2$ away from each other
since at each step we eliminate points which could invalidate this invariant.

Now we need a proof that we can perform this K times, or in other words, we never run out of points potentially in C from which to select at each iteration.

Proof:
We were allowed to assume there exists some set of K points with separation Δ^*.
For convenience, let's call these the optimal points.

Lets examine how many optimal points are eliminated from C in one iteration of the algorithm.
If we select optimal point u, we will see that all the other optimal points are at least Δ^* away from u, and they will not be eliminated from C in this iteration.
If we select non-optimal point u, we will see that at most one optimal point is less than $\Delta^*/2$ away from u.

This due to the triangle inequality. Consider a nonoptimal point u, the nearest optimal point v, and any other optimal point w.

$$\Delta^* \leq d(v, w) \leq d(v, u) + d(u, w)$$

Since v no farther from u than w, $d(v, u)$ may be less than $\Delta^*/2$, but $d(u, w)$ will certainly be at least $\Delta^*/2$.

Therefore at most one optimal point is removed from C in each iteration of the algorithm.

Therefore this algorithm can be run at least K iterations before running out of points to select.

Runtime:
The algorithm runs K iterations, each iteration takes $O(|X|)$ work, so the runtime is $O(K * |X|)$.

(c) Modify the algorithm from B to return NONE if it runs out of points potentially in C that are not already in C.
Observe that the optimum separation delta may only be one of $\binom{|X|}{2}$ values: the distances $d(i, j) : i \neq j$.
Sort the values from highest to lowest, and run the algorithm with these values.
Return the none-NONE answer corresponding to the input with highest Δ^*

The algorithm runs in $O(|X|^2 * log(|X|^2) * K * |X| * |X|^2) \subseteq O(K * |X|^3)$