15-451 Assignment 07

Karan Sikka
ksikka@cmu.edu
Collaborated with Dave Cummings and Sandeep Rao.
Recitation: A
November 7, 2014

1: A Densely-Knit Community

(a) Consider the following assignment of values to x:

Let $x_i = 0$ if v_i not in S^*
Let $x_i = \frac{1}{|S^*|}$ if v_i is in S^*
Let $x_{ij} = \min(x_i, x_j)$

Note that the constraints are satisfied. All values are at least 0. The sum of the x_i is 1 since it's $|S^*| * \frac{1}{|S^*|}$.

$$\texttt{OPT}_{LP} \geq \sum_{i,j \in E} x_{ij} = edges(S^*) * \frac{1}{|S^*|} = OPT$$

(b)

(c)

(d)

2: Large + Dense = Difficult

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

Given a (G, K, D) and a potential solution S of size K, compute the density which is

$$\frac{|\texttt{edges}(S)|}{|S|}$$

Note:Computing edges of S can be done in $|S|^2$ time.

The problem is NP-hard because we can show a a Karp reduction from K-Clique to it.

Given G, K we want to output YES if there exists a K-clique in G.

Observe that the number of edges in a K-clique is $\frac{K(K-1)}{2}$ because you can pair each vertex with each other vertex and divide by 2 to eliminate the same pair written backwards.
Therefore the density of a K-clique is

$$\frac{\frac{K(K-1)}{2}}{K} = \frac{K-1}{2}$$

Also, a graph that is not a K-clique has a strictly smaller density because you would have to add edges to make it a K-clique, thereby increasing the density.
Therefore if a graph has density equal to $\frac{|V|-1}{2}$ it must be a K-clique.

We construct an input to the Reasonably Sized problem as follows:
Let $D = \frac{K(K-1)}{2}$ Let G be the same graph as in the K-clique problem. Let K be the same K.

This correctly outputs YES when there is a K-clique and NO when there isn't because of the way we set D.

Since the problem is NP and NP-hard, it's NP-Complete.

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 there is a Karp Reduction from the Independent Set decision problem which is NP-hard to this problem. 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, and examine how far away all other points are from it.
2. Add u to the set of known points C.
3. Remove u from the set of points potentially in C.
4. Remove all points not at least $\Delta^*/2$ from u from the set of points potentially in 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 the potentially-in-C set in one iteration of the algorithm.
Claim: At most one optimal point is removed from the potentially-in-C set.
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 the potentially-in-C set in this iteration. u will be the only optimal point removed.
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 the potentially-in-C set in each iteration of the algorithm.

We know there were K optimal points to start out with in the potentially-in-C set.

Therefore the 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) You could run the algorithm on all possible values of Δ^*. More formally:

Modify the algorithm from B to return NONE if it runs out of points potentially in C before running K iterations.
Observe that the optimum separation delta may only be one of $\binom{|X|}{2} \leq |X|^2$ 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^*

Sorting takes $O(|X|^2 * log(|X|^2))$ and the rest takes $O(K * |X| * |X|^2)$. The limiting step is $O(K * |X|^3)$

Note: you could do this faster by using binary search instead of linear search, but it doesn't matter for the sake of this problem.