CS F402 Computational Geometry

Programming Project
Euclidean K-Supplier Problem

Raj Kashyap Mallala
2017A7PS0025H

L Srihari
2017A7PS1670H

T Naga Sai Bharath
2017A7PS0209H

June 2020

1 Problem Statement

Given a set $\boldsymbol{F}$ of $\boldsymbol{m}$ points and another set $\boldsymbol{C}$ of $\boldsymbol{n}$ points in the plane, the objective is to open a set $\boldsymbol{F'} \subseteq \boldsymbol{F}$ of $\boldsymbol{k}$ facilities such that the maximum distance of any client in $\boldsymbol{C}$ to its nearest open facility in $\boldsymbol{F}$' is minimized. The distances between the points satisfy two properties: symmetry (i.e. d(u,v) = d(v,u) for all $u, v \in \boldsymbol{F} \cup \boldsymbol{C}$) and triangle inequality (i.e $d(u,v) + d(v,w) \geq d(u,w)$ for all $u, v, w \in \boldsymbol{F} \cup \boldsymbol{C}$).

Let $d(f, \boldsymbol{F'})$ = minimum distance from point f to g where g is any point in $\boldsymbol{F}$'. Then,

$$d(f, \boldsymbol{F'}) = min\ d(f,g) : g \in \boldsymbol{F'} \tag{1}$$

The given problem can be expressed as,

minimize $max\ d(f,\boldsymbol{F}')$, where

$f \in \boldsymbol{C}$ and $\boldsymbol{F'} \subseteq \boldsymbol{F}$ such that $|\boldsymbol{F'}| = \boldsymbol{k}$

2 Introduction

Multiple applications of the euclidean K-supplier problem can be found. One obvious application is in the domain of facility location in supply - chain markets and businesses, commonly observed in large supermarket chains. Here the consumer outlets form the set $\boldsymbol{C}$ and locations of potential intermediate suppliers form the set $\boldsymbol{F}$.
Other applications include opening hospitals in a city with limited capacities, opening of centres for the disposal of obnoxious substances, setting up servers in a network, data mining and information retrieval such as data clustering.

3 About the paper in concern

The implementation for the Euclidean K Supplier problem done in this project is based on the research paper by Nagarajan *et al.*[1]. They present a 2.73 approximation algorithm. It is very close to the nearest possible approximation algorithm (It is NP-hard to approximate the algorithm better than 2.65). The approach taken in this algorithm is also impressive. They use information on the pairs of clients to solve the problem. We have implemented the algorithm for 2 dimensional case.

3.1 Algorithm

As common in many bottleneck optimization algorithms, an additional parameter L is selected (which is the objective value) and outputs one of the following :
1. a certicate showing that the optimal k-supplier value is more than L, or
2. a k-supplier solution of value at most α * L. where $\alpha = 1 + \sqrt{(3)}$ is the approximation ratio.
The final algorithm uses binary search to find the best value of L.

So, for a chosen L, we do the following.
A maximal subset of clients is selected such that the distance between any pair of clients is greater than $\sqrt{(3)}$ * L. We construct a graph based on these selected clients as vertices. We also select a subset of facilities such that there is a one to one correspondence between these facilities and edges in the graph. The facilities in this subset are selected such that there is atleast one and atmost two clients in the graph within distance L.
In other words, there is an edge for a pair of vertices in that graph, if there is a facility which is at distance less than L from both the vertices. For a client, if there is no such partner client satisfying that rule, then there is a self loop for that client in the graph.
More formally, lets say the new graph G has the vertex set P edge set E=$E1 \cup E2$
$E1 = (u,v) : u,v \in P, \exists f \in F$ with $d(u,f) \leq L$ and $d(v,f) \leq L$
$E2 = (u,u) : u \in P, \exists f \in F$ with $d(u,f) \leq L$ and $\forall v \in P$, $(u,v) \notin E1$
Then we find the minimum edge cover on this new graph G. If the number of edges in the minimum edge cover is greater than k, then the optimal value is greater than L else output the facilities corresponding to the edges in the minimum edge cover.

For constructing the graph, we have used an algorithm for finding the approximate nearest neighbour with $(1 + \epsilon)$ approximation ratio ($\epsilon = 0.2$) by Arya and Mount.
And also, for finding minimum edge cover, we find the maximum matching of G using Micali and Vazirani's algorithm and then in linear time find the minimum edge cover of G

3.2 Time Complexity

Generally for constructing the graph, it takes $O(n^2)$ time which involves finding the vertices and then edges. But using approximate nearest neighbours algorithm by Arya and Mount, we can construct the graph in $O(nlogn)$ time complexity but with a loss of $1 + \epsilon$ factor. And the fastest known edge cover algorithm is finding the maximum matching and in linear time find the minimum edge cover. Micali and Vazirani's maximum matching algorithm runs in $O(E\sqrt{V})$. In our case, $E \leq n$ and $V \leq n$. Because every edge has a corresponding facility and therefore edges are less than number of facilities. And also, the number of vertices is a subset of clents and therefore number of vertices is less than total number of clients. Here, n

is number of facilities and clients combined. Therefore, this step takes $O(n^{1.5})$ time.
For finding the minimum edge cover from maximum matching takes linear time. Because we go through the graph once and note the unsaturated vertices (vertices which don't have any edge in maximum matching). Now for every such vertex, we add any of the edge of that vertex to the set of maximum matched edges. In this way, every vertex is covered and we have minimum edge cover.
Considering both the steps of creating the graph and finding the minimum edge cover on it, our algorithm runs in $O(n^{1.5})$ time.

3.3 Pseudocodes and Explanation

Pick a maximal subset $P \subset C$ such that each pairwise distance in P is more than $\sqrt{3}.L$;
Construct graph G=(P,E) with vertex set P and edge set $E = E1 \cup E2$, where
E1 = (u,u) : $u \in P, \exists f \in F$ with $d(u, f) \leq L$ and $d(v, f) > L \forall v \in P \backslash u$;
E2 = (u,v) : $u, v \in P, \exists f \in F$ with $d(u, f) \leq L$ and $d(v, f) \leq L$;
Compute the minimum edge cover $T \subset E$ in G;
if $|T| > k$ **then**
  the optimal value is larger than L;
else
  output the facilities corresponding to T as solution;
end

Algorithm 1: Algorithm for Euclidean k-supplier

As a part of this algorithm, we are required to calculate the minimum edge cover. We achieve this by first finding the set of edges in the maximum match and then for each vertex of degree zero, add any of its edges to the set. It can be shown that this effectively converts the output of maximum matching algorithm ([2]) to a set of edges in minimum edge cover, as required by the Euclidean k-Supplier algorithm.

$M \leftarrow$ output of Maximum Matching on Auxillary graph;
for *each vertex v of degree 0* **do**
  Pick any edge e connected to v;
  Add e to M;
end
Output M;

Algorithm 2: Minimum edge cover from Maximum Matching

$P \leftarrow \emptyset$ and $P' \leftarrow \emptyset$;
for $v \in C$ **do**
  $v' \leftarrow$ approximate nearest neighbor of v in P';
  if $d(v, v') > \sqrt{3}(1+\epsilon)^2 L$ *or v' = NIL* **then**
    $P \leftarrow P \cup v$ and insert v into P';
  end
end

Algorithm 3: Algorithm for computing vertices

Construct data structure P' containing points P, and initialize $E \leftarrow \emptyset$;
for $f \in F$ **do**
  $u \leftarrow$ approximate nearest neighbor of f in P';
  $v \leftarrow$ approximate second nearest neighbor of f in P';
  if $d(u, f) \leq (1+\epsilon)L$ *and* $d(v, f) > (1+\epsilon)L$ **then**
    set $E \leftarrow E \cup (u, u)$ and label edge (u,u) by facility f.;
  end
  if $d(u, f) \leq (1+\epsilon)L$ *and* $d(v, f) \leq (1+\epsilon)L$ **then**
    set $E \leftarrow E \cup (u, v)$ and label edge (u,v) by facility f.;
  end
end
Output E;

Algorithm 4: Algorithm for computing Edges

4 Results

The $(1 + \sqrt{3})$ approximation Euclidean k-Supplier algorithm proposed in [1] has two major sub-routines, namely, An algorithm for finding maximum matching in general graphs [2] and An optimal algorithm for approximate nearest neighbor searching in fixed dimensions (Sunil Arya and David M. Mount [4]), which are by themselves well researched in the algorithmic domain. In this programming assignment, the first major subroutine on Maximum Matching, also known as the MV algorithm, has been programmed from the scratch based on the algorithmic exposition given by Paul A. Peterson and Michael C. Loui [3]. The second major subroutine on Approximate nearest neighbors has been implemented with the aid of the ANN library provided for public use by the authors themselves [5]. The complete coding was done in C plus plus programming language. As the performance and correctness of the subroutines are crucial for efficient working of the main algorithm, we provide the results of unit tests on these sub routines first. The unit tests are followed by results of the complete algorithm.

4.1 Unit test on component Algorithms

Unit tests for Maximum Matching algorithm was conducted with input from NetworkX python library. The library's builtin max_weight_matching method was used to test the results of our code. It was observed that the number of matchings were same for all 36 unit tests conducted. The running time of our algorithm was found to be faster for large size of input graphs. 16 of the different varieties of unit tests have been tabulated in the table 4.1. The choice of unit tests were based on the variety in the type of graph for the first 11 tabulated entries and the remaining were tested for large size of input. The input and output corresponding to each of these entries have been documented in an accompanying text file.
Since, the nearest neighbors identified by [4] are not easily verifyable because of their approximate nature (there is no single correct answer in many cases), we refrain from testing the algorithm for its correctness. Also, since the implementation has been taken from a public and well documented library, it is expected that the correctness would have been established by the authors themselves. Further, this algorithm is based on the research paper [4] as ground truth, hence the algorithm's time complexity need not be tested because of the fact that the library was created to perform with the proven $O(n^2)$ worst case complexity and $O(log^3 n)$ expected time. The following subsection provides the unit tests on the maximum matching implementation by us.

4.1.1 Maximum Matching Algorithm (MicaliVazirani.cpp)

Unit test results					
Type of graph	Number of nodes	Number of edges	Running Time (microseconds)	Size of NetworkX output	Size of our output
Single Bloom, Two extensions	7	7	996	3	3
Two Blooms, Three connecting edges	12	13	990	6	6
Two blooms, three connected components	10	15	995	5	5

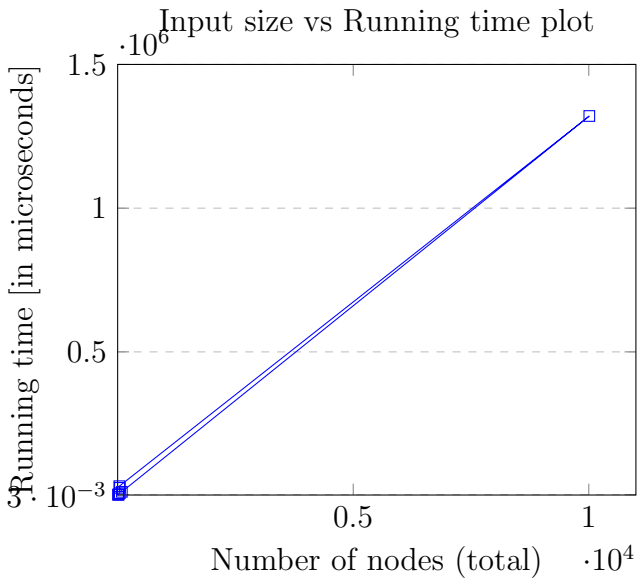
Two blooms, one embedded inside other	9	11	997	4	4
Two blooms, one embedded , connected at centre	9	11	999	4	4
Three blooms, three connections	16	18	997	7	7
Three hexagons, three bridges, Two extensions	19	21	974	9	9
Peterson graph	10	15	309	5	5
Bull graph	5	5	1019	2	2
Dodecahedral graph	20	30	944	10	10
Frucht graph	12	18	997	6	6
Medium size 2D graph	100	180	5018	50	50
Medium barbell graph	30	101	1995	15	15
Large Barbell graph	121	2472	42883	60	60

Large ladder graph	2000	2998	80748	1000	1000
Large Erdos Renyi Random graph	500	74716	1056175	250	250

4.2 Results of the main algorithm

Unit test results					
Type of graph	k- value	Number of Client nodes	Number of Facility nodes	Running Time (microseconds)	Best L value
Near circle clients embedded facilities	4	10	3	4148	4.17163
Single Line clients Spread facilities	2	6	4	2697	3.29688
Annular Facilities and clients	1	9	14	3718	2.53125
Random many clients few facilities	2	43	3	8466	7.125
Random many facilities Few clients	2	5	30	32678	4.90128
Large grid of clients and few facilities	2	10000	11	1320631	35.0703

Rainbow arc of client and facilities	3	75	14	13146	2.53125
Alternate rainbows of client and facilities	10	27	21	26934	1.09659



5 Conclusions from the empirical study

So in this project, we have successfully implemented the Euclidean k-supplier problem which is one of the topics actively researched in Computational Geometry. We have implemented it based on the research paper by Nagaran *et al.*. We could achieve the Time Complexity of $O(n^{1.5})$ with the approximation factor of $1 + \sqrt{3}$. This algorithm is different in the sense that it used many different algorithms and concepts to arrive at a better approximation algorithm for a very useful problem.

While implementing the algorithm, we faced difficulty in finding the correct value of L. If we choose a very small L, the graph G contains isolated vertices. But for finding minimum edge cover, we should have a connected graph. If L is large, we only get a few vertices and it will not be the optimal L.

Further extensions to this project could be to add a visualization of the clients, supplier and suppliers chosen.

In the course of this project, we have learnt many things from installing CGAL and also the library ANN by Arya and Mount. We have learnt how to install c++ libraries by downloading include and .cpp files and then compiling them and making the .lib and .dll files making available for the compiler to process. It is a new experience for us and also learnt many things about how libraries work.

6 Appendix: Type of input and expected output for each enclosed cpp file

6.1 MicaliVazirani.h

Input: An object of the class MicaliVazirani must be instantiated by passing an adjacency list in the form of $map < int, vector < int >>$ to the construtor. Then the method general_matching_algorithm can be called on that object.

Output: The function general_matching_algorithm returns a $map < pair < int, int >, pair < int, int >>$. If the input had was of single dimension then the second element of pair is taken as '0'.

6.2 euclideanKSupplier.cpp

Environment variables for ANN:
CPLUS_LIBRARY_PATH = <path to include files>
LIBRARY_PATH = <path to library containing .a files in Linux or .DLL and .lib in Windows>

Compilation: g++ euclideanKSupplier.cpp -lANN

Input: Input is through command line.
./a.out (k) (clients file) (facilities file) [in Linux]
./a.exe (k) (clients file) (facilities file) [in Windows]

Output: The best possible L value and the facilities corresponding to that value of L

References

[1] Viswanath Nagarajan, Baruch Schieber, and Hadas Shachnai *The Euclidean k-Supplier Problem*. Integer Programming and Combinatorial Optimization. IPCO 2013.

[2] Silvio Micali, Vijay V. Vazirani *An algorithm for finding maximum matching in general graphs*. 21st Annual Symposium on Foundations of Computer Science (sfcs 1980)

[3] Paul A. Peterson, Michael C. Loui *The general maximum matching algorithm of micali and vazirani*. Algorithmica 3, 511–533 (1988). https://doi.org/10.1007/BF01762129

[4] Sunil Arya, David M. Mount *Approximate nearest neighbor queries in fixed dimensions*. SODA '93: Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms, January 1993

[5] Sunil Arya, David M. Mount *ANN: A Library for Approximate Nearest Neighbor Searching*. http://www.cs.umd.edu/ mount/ANN/