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

3.1 Pseudocodes and Explanation

while *not at end of this document* **do**
  read current;
  if *understand* **then**
    go to next section;
    current section becomes this one;
  else
    go back to the beginning of current section;
  end
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

4.1.2 Approximate nearest neighbour search

4.2 Results of the main algorithm

5 Conclusions from the empirical study

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/