Finding Top-k Influential Set in Directed Graphs

Student Name: Pankaj Sahu

IIIT-D-MTech-CS-DE-12-044
July 30, 2014

Indraprastha Institute of Information Technology
New Delhi

Thesis Committee
Dr. Vikram Goyal, IIIT Delhi (Chair)
Dr. Debajyoti Bera, IIIT Delhi (Chair)
Dr. Somitra Sanadhya, IIIT Delhi
Dr. Anand Gupta, NSIT Delhi

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Data Engineering

Keywords: R*k*NN Query, Collaboration Network, Social Networks, Directed Graph

Certificate

This is to certify that the thesis titled "**Finding Top-k Influential Set in Directed Graphs**" submitted by **Pankaj Sahu** for the partial fulfillment of the requirements for the degree of *Master of Technology in Computer Science & Engineering (Data Engineering)* is a record of the bonafide work carried out by him under my guidance and supervision in the Data Engineering group at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Dr. Vikram Goyal
IIIT Delhi

Dr. Debajyoti Bera
IIIT Delhi

Abstract

A RkNN (Reverse k Nearest Neighbor) set of a query q is known as the influence set of q that contains the top k influential points. This query has got a considerable amount of attention due to its importance in various applications involving decision support system, profile based marketing, location based services, etc. Although this query has been largely studied in Euclidean spaces but there is very less work done in the context of large graphs. In this dissertation, a framework has been proposed for RkNN query over directed graphs. We present a heuristic that cuts out the search space substantially for finding out RkNN of a query point. To the best of our knowledge there has been no work done for RkNN query over directed graphs. We conduct extensive experiments over some real world data sets like DBLP, social network, product co-purchasing network of Amazon and study the performance of the proposed heuristic in various settings. Experiments show the effectiveness of the proposed heuristic. We also study a co-authorship application in the context of RkNN query over undirected graph, wherein we design a metric to define the similarity between two authors. Here RkNN query can be interpreted as a query to find out the influence set of an author from an authors collaboration network (DBLP).

Acknowledgments

I dedicate this thesis to my parents and maternal uncle. Their support and unwavering confidence in my ability helped me to achieve my academic dreams.

Firstly, I would like to express my deepest gratitude to my advisor Dr. Vikram Goyal and Dr. Debajyoti Bera for their guidance, encouragement and continuous support during my thesis. I am very grateful to have got the opportunity to work under them. Their guidance helped me throughout the research work and without it; this thesis work would not have been possible.

Besides my Advisor, I would like to thank Sakshi Tiwari and Prachi Agrawal for their help and support during this work. My sincere thanks also goes to Anuj Rajani for spending his valuable time to review my thesis.

I wish to express my gratitude to Dr. Pankaj Jalote and all other associate members of IIIT-Delhi for providing me such a brilliant platform to enhance my knowledge.

Also, I would like to thanks to all my friends in IIIT-Delhi who made my stays dearly chrisable and memorable especially, Rohit(Romley), KK, Anuj, Rohit(Jain), Gajendra, Sandipan, Aritra, Noufal, Ganesh, Nishant, Veeru and Prateek.

Last but not the least, I would like to thank Dr. Somitra Sanadhya and Dr. Anand Gupta for their acceptance to be a part of my thesis committee.

Pankaj Sahu
Indraprastha Institute of Information Technology, New Delhi

Contents

List of Figures:

List of Tables:

Chapter 1

Research Motivation and Aim

1.1 Motivation:

The elementary problem that arises in several marketing and decision supporting systems is to determine the "influence" of a data point on the database. The concept of influence is often difficult to formalize. For example, influence of a new restaurant outlet. We first develop an intuitive notion of influence sets through examples to motivate our formalization of it.

Example:
Consider that Company **X** opens its new restaurant in a particular location. The **X's** simple endeavor is to find the chunk of its customers that would be more likely to use this facility. Alternatively, one may require to find the chunk of customers of Company **Y** who might find the new restaurant of the Company **X** more convenient than the locations of **Y**. Such chunk of customers is roughly what we call influence sets. Let us now make the notion of an influential set more precise. Suppose a data set P and a query point q is given, the goal is to find the subset of points in P influenced by q.

One way is to solve the above defined problem statement is using the well known concept of Nearest Neighbor (NN) query or K-NN in which one returns the k nearest neighbors of a given query point. The second way is to use the Euclidean Distance concept, defining an epsilon radius and returning all the points within that radius.

In Euclidean space, the distance between any two objects is directly calculated by their comparative place in the space. However, in practical scenarios, objects can follow some pre-defined set of path of an underlying network where the Euclidean distance is inapplicable. In figure 1.1 Euclidean distance between point's q and p_4 is less, but the actual path shows that point p_4 is far as compared to other points p_1 and p_2. Thus, the important measure for spatial network is Network distance. Network distance of any two objects or nodes follows the shortest path of those two objects or nodes, rather than their Euclidean distances. But these methods fail in giving the influential sets because NN queries are not symmetric.

For instance, if a point p is NN of query point q, it is not necessary that q is also NN of p i.e. there may be some other point r which is NN of p. For example, in figure 1.1, P_1 is the nearest neighbor of query point q but this relation is not bidirectional, instead point P_2 is the NN of P_1. So, we use the concept of Reverse Nearest Neighbor query (RNN) to take into account the above mentioned issue.

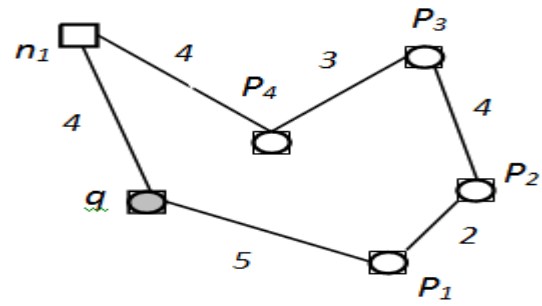


Figure 1.1: Euclidean Distance over Network Distance

Reverse Nearest Neighbor Query (RNN):

If a dataset P contains a query point q and another a data point p then the (monochromatic) query point q's reverse k nearest neighbors (R*k*NN) return all those data points p∈*P* that have q as one of their k nearest neighbors (*k*NN).

$$\mathbf{R}k\mathbf{NN(q)} = \{p \in P \mid \mathbf{dist}(p, q) \leq \mathbf{dist}(p, p_k(\mathbf{p})), \text{ where } p_k(\mathbf{p}) \text{ is the } k^{\text{th}} \text{ NN of } p\}$$

Similarly, given two datasets P and Q and a query point q, the (bichromatic) query point q returns all data points $p \in P$ which are nearer to q than any point of $q_i \in Q$.

$$\mathbf{bR}k\mathbf{NN(q)} = \{p \in P \mid \mathbf{dist}(p, q) \leq \mathbf{dist}(p, q_k(\mathbf{p})), \text{ where } q_k(\mathbf{p}) \in \mathbf{Q} \text{ is the } \mathbf{k^{th}} \text{ NN of p}\}$$

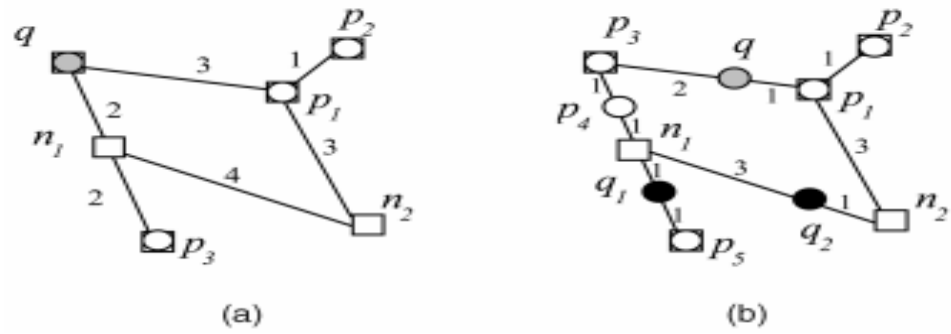


Figure 1.2: RNN queries in the graphs. (a) For monochromatic query. (b) For bichromatic query.

RNN queries differ based on their application environment. In P2P network where every point only resides on a node, it may be possible that a few nodes do not contain a point, which is mostly known as restricted network. In above figure 1.2(a) let's say a DBLP co-authorship (collaboration) network is represented where author q wants to collaborate with another author in some specific field. A (monochromatic) RNN query will return any one author whose NN is q, from all of the existing same specific field authors {P_1, P_2, P_3}. Here nodes n_1 & n_2 represent the authors which are not working in the specific field, which query author demanded, thus, in this case these are irrelevant for the query author. In above figure 1.2(a), according to the edge weight RNN (q) = {P_3}, because P_3's NN is q.

The other network where point (data & query) can reside anywhere on the edge of a graph, is called an unrestricted network. Figure 1.2(b) represents an example of bi-chromatic query in a road network,

where some residential blocks P & restaurants Q are represented by points P_1 to P_5 and q_1, q_2 respectively. Nodes n_1 & n_2 is the road junctions that do not contain any residential blocks or restaurants. In figure 1.2(b), an interesting bi-chromatic query is a new restaurant q wanting to check if location is perfect in terms of getting more customers as compared to its rival restaurants q_1 & q_2. In the figure, on basis of their edge weights, RNN (q) = {P_1, P_2, P_3}. If RNN (q) is compared to its rival restaurants i.e. RNN (q_1) = {P_4, P_5} & RNN (q_2) = {Ø} then RNN {q} can be seen to have gotten the perfect location in terms of getting more customers.

All the above examples discuss only the single RNN retrieval. If any query point requests for R*k*NN query retrieval then that will return all the data points which contain that query point in their *k*NN. In above graph 1.2(a), a (monochromatic) RkNN query for k=2, R2NN (q) = {P_1, P_2, P_3} where every point p_1 to p_3 contain a q in their 2NN. In graph 1.2(b), a (bichromatic) bRkNN query for k=2 then bR2NN (q) = {P_1, P_2, P_3, P_4}, bR2NN(q_1) = { P_4, P_5} and bR2NN (q_2) = {P_1,P_2,P_5}.

Earlier a lot of work has been done on weighted undirected graphs. We have proposed a technique to calculate the influential set on a weighted Directed Graph. Assigning weight on the graphs is a major problem in large Networks. The weight w(n_i, n_j) is determined by the application domain, like in a road network, it may be defined by travel time or distance between the two segments connecting n_i and n_j, whereas in some another network or graph, it defined by different parameters according to their application.

In this thesis, we propose two algorithms for directed graphs. For an undirected graph, we have used and have implemented already described RNN algorithm and their optimized version in [1]. All the algorithms results are verified through the naïve approach. In this thesis, we have only worked on monochromatic R*k*NN query for the restricted network. Here we mainly focus on some major P2P application (co-authorship network, Social network), but have also experimented for few other applications.

1.2 Problem Description:

A directed graph or digraph is represented as G (V, E), where V refers to a set of vertices or nodes n and E represents the ordered pairs of vertices known as arcs or directed edges, i.e. e (n_i, n_j) if n_i is connected to n_j in the direction of n_i to n_j, where n_j is called the head node and n_i is called the tail node.

In [1], all the work has been done for undirected graph where weight of edges is same for the both the directions, i.e. w (n_i, n_j) = w (n_j, n_i). But the same does not hold true for directed graphs unless and until both the nodes are connected to each other with the same weight. Therefore, the network distance d is not symmetric for the digraph i.e. d (n_i, n_j) ≠ d (n_j, n_i) and that's the reason the lemma which was proposed by Papadias et al. [1] is not applicable for digraph.

The another problem is in the particular application of an undirected graph which is a co-authorship network. We use the term co-authorship and collaboration, interchangeably. In co-authorship network, node represents author and an edge between two authors exists if they have published one or more papers together. But assigning the weight (strength measure) or nearness (closeness) between the two authors is a complicated task. [1] experimented on this application and assigned the weight to be 1 that does not work for every query in terms of checking the nearness between the authors.

For assigning the closeness or weight between two authors, we require the two conditions to satisfy:

- ❖ Number of times, they collaborated for the paper.
- ❖ How much, they contributed or interacted during their collaboration for the paper.

1.3 Research Contributions:

Our contributions in this work are as follows:

- We propose a framework for the Reverse Nearest Neighbor Query in Directed Graph. We also provide a proof of correctness for our proposed framework.

- We propose two methods, namely, Directed Eager (D_M) and their optimization once Directed Eager Materialization (D_EM) for performing monochromatic reverse nearest neighbor query in the directed graph for dynamic value of k (requested points).

- We present a novel idea for getting the weight factor to present the relationship and nearness (closeness) between two authors in the collaboration network.

- We conduct extensive experiments on both directed and undirected graphs and compare the performance of the algorithm through various settings.

Chapter 2

Related Work

2.1 RNN Query for undirected graph:

A great amount of work in spatial domain has been done through the spatial queries, especially applications those involving in the process of decision support system, profile based marketing, location based service system. Spatial queries commonly address two groups of frameworks, based on: 1) Euclidean distance 2) network distance.

Existing work on the first group of framework [9, 10] considers Euclidean (Cartesian) spaces, where the distance between any two objects is directly calculated by their comparative place in space. Although it's possible in a real scenario that no path exists between the two objects. Yet, in practical scenarios, objects can stick with more or less pre- determined set of path of a spatial network where the Euclidean distance is regarded to be impractical. So the network distance is the necessary parameter for determining distance on spatial networks.

The second group of framework [1, 11, 12, 13] considers spatial network, where the queries are only solved by the network distance. Network distance of any two objects or nodes follows the shortest path between those two objects or nodes, rather than their Euclidean distances.

Papadias et al. [1] are the first to suggest the solution for all forms of RkNN queries in spatial networks. They proposed a lemma to prune the search area and to calculate the influential set effectively.

Lemma proposed by [1]:
"Let q be a query point, n a graph node, and p a data point satisfying d (q, n) > d (n, p). For any point $p' \in P$ whose shortest path to q passes through n, then d (q, p') > d (p, p'), i.e., $p' \notin$ RNN (q)."

On the basis of this lemma, Papadias et al. [1] proposed two alternatives to exploit NN-search which were called range-NN and verification queries. A *range-NN(n, k, e)* query accessed the k closest data points with (network) distance less then e from node n, if k points exist. Otherwise, it returns less then k point (sometime possibly 0). For example, in fig. 1.2(a), if k=1, n=n_1 and e=2 then range query returns no result because d(n_1, q) = d(n_1, p_3) = 2≥e.

Similarly the verification query, *verify(p, k, q)* checks in kNN's of point p containing a query q, by applying range query on the node that contains point p. *Verify(p, k, q)* is equivalent to *range-NN(p,k,d(p,q))* and search terminate when q is found but the distance d(p, q) consider here is the maximum.

On the basis of Lemma and two alternatives given above Papadias et al. [1] proposed two algorithms Eager (E) and Lazy (L) Algorithms to calculate the RkNN for a given query point q. Later, optimization of these two algorithms eager materialization (EM) and lazy-extended pruning (Lazy-EP) were also

proposed. But the framework they have proposed only for the undirected graph. In this work, we extend it to make it applicable for the directed graph.

Safar et al. [11] used a network Voronoi diagram (NVD) for efficiently processing the RNN queries in spatial networks. The NVD uses the Voronoi cell that has the nodes & the edges which are nearer to the generator point of the cell compare to other points in the network. In their follow-up work [12], they expand their technique to process R*k*NN queries and reverse k farthest queries in spatial networks.

Cheema et al. [13] are the one to demonstrate a continuous R*k*NN monitoring algorithm for moving objects and queries in spatial networks. They cover both the spatial network directed and undirected However, the technique which they used is different and applied so many lemmas causing the verification query to get expensive. Also, there is no optimization technique, they performed.

2.2 Weightage scheme on Collaboration Network:

The scientific community is growing & exploring through research article every day. Scientists or researchers play an important role in this community, who collaborate with other scientists for their research work. On the basis of their collaboration, scientific community creates co-authorship or collaboration network. In this network, any two authors are connected only if they have published one or more papers together. Interaction among the scientist cover fairly larger phase of communication before their collaboration. In their communication phrase, scientists do not only read, write & discuss, but they also get to understand each other better in their area of work before collaborating on any paper. Influence factor not only covers communication, but also other factors that are important for research collaboration. A literature survey [3] described the internal and external influence factor of the researcher during their research collaboration.

Papadias et al. [1] experimented for DBLP co-authorship graph for ad-hoc queries and for that they assigned a weight between two authors (node) just to be 1 which is not useful for retrieving the influential co-authors of query author. Related work [2] was done in this area, but the provided strength measure was not sufficient to predict the influence. So, we have proposed an efficient formula to calculate the weight factor for each edge in the graph which defines the relationship or closeness between the authors.

Chapter 3

Proposed Framework

3.1 Directed graph:

The proposed algorithm solves the R*k*NN query for directed graph (digraph). An R*k*NN of query point q returns all *k* data points *p*∈*P* which contains q in its one of the k nearest neighbors (*k*NN) where k is an arbitrary positive real number whose value much less than the number of data points in the network.

If a query point q is finding R*k*NN data points in digraph then the reverse nearest neighbor query returns *k* nearest data point whose arc direction reached to q in terms of their network distance. In a large digraph is difficult to determine for all the nodes those arc directions are reaching to query node and to solve it efficiently there is a need to maintain two graphs. One is the Main graph G_M and the other one is their Reverse graph G_R (reverse direction of edges of main graph G_M). The proposed algorithm is processing all the tasks in-memory and there is no other storage method we are using in this work. The proposed algorithm is similar to Dijkstra and uses the min-heap H (priority heap) to extend the network around the query q.

A simple naïve approach is to check, all data points *k* nearest neighbors and if any point contains a q as one of their *kNN's* then that point is included to the R*k*NN set of q, but in large network where the number of data points are more, it takes a lot of time.

In another approach, traverse the reverse graph G_R from the query point q, and every encountered node n in G_R is inserted into the heap H. If the node n (deheaped from the H) contains a point *p*∈*P* apply a NN query for p on main digraph G_M. If a q is one of *kNN's* of p then *p*∈R*k*NN*(q)*. But, the same kind of problem is happening in this approach also because the RNN set is not fixed and it accesses the most of the network. So to minimize extension of the network, following lemma for digraph have been proposed, which states:

Lemma:
In a directed graph, a query point q, a data point p and a node n, if d(n, q) > d(n, p) is satisfied and any other data point p′≠ p whose shortest path to q, passes through n, then point p′ is not in the RNN set of q, i.e. p′∉ RNN(q), because d(p′, p) < d(p′, q).

Proof: In digraph distance is not symmetric.

$$d(p', q) = d(p', n)+d(n, q) > d(p', n)+d(n, p) \geq d(p', p) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

According the lemma, point p′ is nearer to point p and not to query point q, that's the reason p′∈ RNN(p) and not the RNN(q). In this situation node should be pruned and not to be expanded.

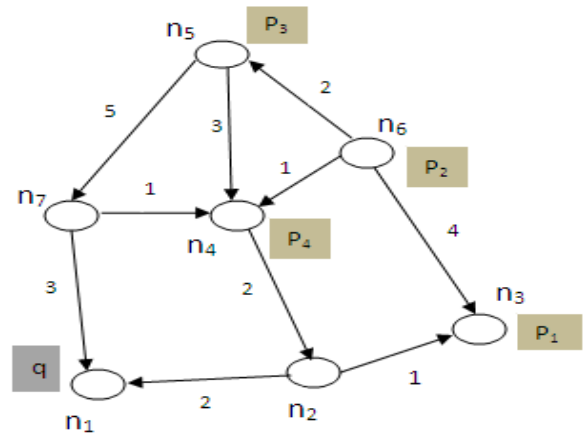
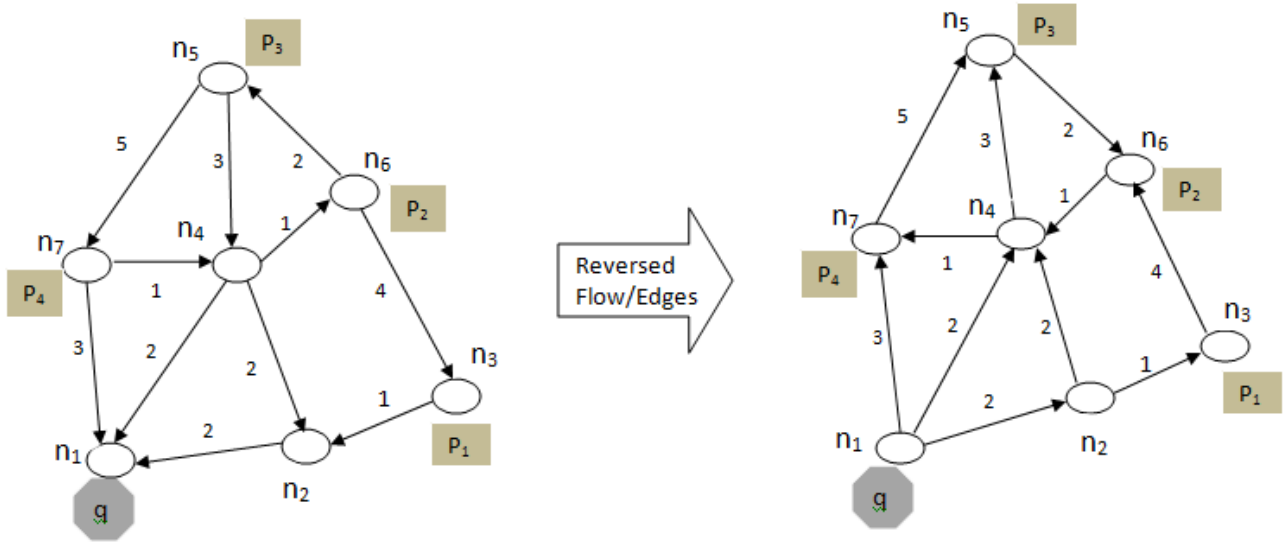


Figure 3.1: Lemma condition on directed graph

In figure 3.1, d (n_2, q) = 2 > d (n_2, p_1) = 1. According the lemma, any data point (in figure 3.1 p_4) whose shortest path to q; passes through n_2 can't be the RNN of q as the data point p_4 is nearer to p_1. In this case node n_2 will be pruned and not to be expanded further. In this network there is no point in the RNN set of q but RNN (p_1) = {p_4}, RNN (p_4) = {p_2} and p_3 also does not have any point in the RNN set.

On paper [1] have proposed two alternatives of the NN query that is also applicable for any given directed main graph G_M. Two alternatives are range-NN query and verification query. A range query, *range-NN (n, k, e)* returns k nearest data point's whose network distance lies within range e from node n. If k exists, otherwise less than k point. In figure 3.1, if n=n_2, k=1, e=2 then range-NN query return point p_1 whose distance 1<e. Similarly the verification query, *verify (p, k, q)* checks in kNN of point p containing a point q by applying range query on the point p. *Verify (p, k, q)* is equivalent to *range-NN (p, k, d (p, q))* and search terminate when q is found but the distance d (p, q) consider here is the maximum.

With the help of these two NN alternatives and using main digraph G_M as well as reverse digraph G_R, the following algorithm has been proposed.



(a) (b)

Figure 3.2: Example of the directed graph (a) Main graph G_M. (b) Reverse graph G_R.

3.1.1 Directed Eager (D_E) algorithm:

Directed eager to some extend is similar to the Eager algorithm of previous paper [1], but there are some major differences. D_E algorithm is using a two graph G_M and G_R. The alternatives of the NN, the range and the verification query are applied on the main digraph G_M. All other operations are done on the reverse graph G_R. D_E algorithm expands the directed network on a query point q around in reverse graph G_R. If q finds any node n (not containing a point) then it runs the *range-NN(n, k, d (q, n)) query* on main digraph G_M around node n which returns *k*NN(n). If *k*NN(*n*) less then *k* then adjacent node n_i of n in reverse graph G_R inserted into heap H, otherwise node n will be pruned and will not be expanded further because all, the point in *k*NN(n) whose distance d(p, n)<d(n, q) and that's the case n (according the lemma) cannot lead to the RNN of q. Similarly, if q finds any node n_p (containing a point p) it runs the verification query; verify*(p, k, q)* on main digraph G_M around point p and it returns *k*NN (p). If $q \in k$NN(p) then p is added into RNN set of q. If *k*NN(p)<k then adjacent node n_i of n_p (where n_p edges going) in the reverse graph G_R inserted into heap H, otherwise node n_p will be pruned and will not be expanded further.

Algorithm D_Eager (q, k)

```
1. insert <n(q), 0> into H      // n(q) is the node containing query point q
2. while(not-empty(H))
3.         <n, d (q ,n)>:= de-Heap(H)
4.         if n is not visited before
5.             mark n as visited
6.             if(n not contain a point)
7.                     kNN(n) = range-NN(n,k,d(q,n)) in main digraph G_M.
8.             else if (n contain a point p)
9.                     kNN(p) = verify(p,k,q) too check in main digraph G_M.
10.                    if q discovered by verification add p to RkNN(q)
11.            if(|kNN(n)|<k or |kNN(p)|<k)
12.               for each non-visited adjacent node n_i of n in reverse digraph G_R
13.                    insert <n_i, d(q,n)+w(n,n_i)> into H
14. return RkNN(q).
```

In figure 3.2, RNN query of q for *k*=1, D_Eager algorithm first initializes the heap H, starts from node n_1 (source node which contains a query point q) and insert <n_1(q), 0> into H. After that, first deheaped node n_1 , doesn't find any point (through *verify* query in G_M) then its adjacent nodes <n_2, 2> <n_4, 2> and n_7, 3> in reverse graph G_R are inserted into H. The next de-Heap node n_2 (do not contain a point) faces a range-NN (n_2, 1, 2) on main graph G_M that doesn't return point (KNN (n_2) =∅) within range distance 2. So, the node n_2 adjacent node <n_3, 3> in G_R is inserted into H. The subsequent de-Heap node n_4 (too, do not contain a point), too faces a range-NN (n_4,1,2) in G_M and return a nearer point p_2, because d (n_4, p_2) =1<d (q, n_4) =2. In this case node n4 (according the lemma) to be pruned and their adjacent nodes will not be inserted into H. Next node n_7 is de-Heaped (contain a point p_4), faces a verify (p_4, 1, q) in G_M that return nearest point p_2 and search terminate. At last node n_3 (contain point p_1) in H removed and finds q their NN (through *verify (p_1, 1, q)* query in G_M). So p_1 added into RNN set of q. After that H is empty means all the point is explored near around the query q. Final result contains RNN (q) = {p_1}.

3.1.2 Materialization for the directed_eager:

We also optimized the D_Eager algorithm through the help of materialization. Materialization method pre-compute the shortest distance between nodes and access the network information in term of network distance in constant time. A simple naïve approach for materializing the information is to apply the *k*NN query on each node, but the problem is, it accessed many node multiple times and that is not efficient for large graph. So the paper [1] proposed all-NN algorithm that accessed the network only once. We used this information in similar way of paper [1] expect some changes for the digraph in All-NN algorithm. So the proposed directed All-NN algorithm is as follows-

Algorithm Directed_All-NN (k)

1. for each node n those contains point in main digraph G_M
2. insert <n,p,0> into H
3. while(not-empty(H))
4. <n,p,d(p,n)>:= de-Heap(H)
5. if |kNN(n)|<k and p ∉ kNN(n)
6. add <p,d(p,n)> to kNN(n)
7. for each adjacent node n_i of n in reverse graph G_R
8. if|kNN(n_i)|<k and p ∉ kNN(n_i)
9. insert <n_i, p, d(p,n)+w(n,n_i) > into H
10. return kNN

In figure 3.2 materialization for single NN, first initializes the heap H and then insert all the nodes of main graph G_M those contain point $p \in P$ with distance zero into heap H, i.e. <n_3,p_1,0>, <n_6,p_2,0>, <n_5,p_3,0>, <n_7,p_4,0>, <n_1,q,0>. Next, first de-Heaped node from the H; is <n_3, p_1, 0>, add NN(n_3) = {p_1} and insert the adjacent node into H from reverse graph G_R which is: <n_6, p_1, d(p_1, n_3)+w(p_1, n_6)>. Similarly, after deheaping node <n_6, p_2, 0>, <n_5, p_3, 0>, <n_7, p_4, 0> and <n_1, q, 0>, the updated NN lists are NN (n_6) = {p_2}, NN (n_5) = {p_3}, NN (n_7) = {p_4}, NN (n_1) = {q} and an adjacent node from the reverse graph G_R in heap H is as following:-

$$
\begin{aligned}
H \;=\; & \langle n_4,p_2,d(p_2,n_6)+w(p_2,n_4)\rangle \equiv \langle n_4,p_2,1\rangle,\\
& \langle n_2, q, d(q,n_1)+w(q, n_2)\rangle \equiv \langle n_2,q,2\rangle,\\
& \langle n_4, q, d(q,n_1)+w(q, n_4)\rangle \equiv \langle n_4,q,2\rangle,\\
& \langle n_6,p_3,d(p_3,n_5)+w(p_3,n_6)\rangle \equiv \langle n_6,p_3,2\rangle,\\
& \langle n_7, q, d(q,n_1)+w(q, n_7)\rangle \equiv \langle n_7,q,3\rangle,\\
& \langle n_6,p_1,d(p_1,n_3)+w(p_1,n_6)\rangle \equiv \langle n_6,p_1,4\rangle,\\
& \langle n_5,p_4,d(p_3,n_7)+w(p_3,n_5)\rangle \equiv \langle n_5,n_4,5\rangle.
\end{aligned}
$$

The subsequent processing of every deheaped node from H, add the NN point if hasn't met the k point before. In above case node n_4, n_2 is deheaped from H and add NN (n_4) = {p_2}, NN (n_2) = {q} respectively. The other deheaped node did not update their NN list because they have already found a single point before.

The directed eager materialization (Directed_EM or D_EM) utilized this materialized information in the algorithm in constant time. Whenever any node n (not contain point) is deheaped it directly retrieves the *k*NN of nodes, instead of range query. If *k*NN of n contains less point within range, then adjacent nodes of n in reverse graph G_R inserted into H. Similarly, if node n_p contain a point p then instead of applying verify

query, it directly retrieves the *k*NN of n_p. For verification of n_p it checks if d (q, n_p) ≤ d (p, p_k (n_p)), (where $p_k(n_p)$ is the k^{th} NN of n_p) then p added to RNN(q) otherwise verify(p, k, q) query applies on point p in main graph G_M.

3. 2 Weightage scheme on Collaboration Network:

Collaboration (Co-authorship) network is an application of undirected weighted graph, where the author is as considered a node and the two authors are linked by an edge if they published one or more paper or document together. The weight of the edge is decided on the basis of utilization of the application. In our work we are interested in finding the most influenced authors of any query author. To calculate this, edge weight represents the closeness or nearness between two authors. So the proposed weight is:

$$W(n_i, n_j) = \frac{1}{e^{\sum_{i=1}^{t}\left(\frac{1}{A(d)}\right)}}$$

Where:-

d = A research document/paper where author n_i and n_j collaborated.
A(d) = Number of authors along with n_i and n_j those collaborated in d.
t = Total number of d (research document) where author n_i and n_j collaborated.

The weight shows that when two authors interacted more during their collaboration then their final weight is less, indicating more closeness comparatively to other two authors who have interacted less in their collaboration tenure. In here we are taking two authors' contribution is directly proportional to their interaction, during collaboration.

Let three papers d_1, d_2 and d_3 published by some authors Ai, where i≥1 i.e. {A1, A2, A3, A4} ∈ d_1, {A2, A3, A5} ∈ d_2 and {A1, A6} ∈ d_3. Then weight between the two authors is as follows:

W(A1, A2) = W(A1, A3) = W(A1, A4) =W(A2, A4) = 1/e^ (1/4) ≡ 0.77
W (A2, A5) = 1/e^ (1/3) ≡ 0.71
W (A1, A6) = 1/e^ (1/2) ≡ 0.60
W (A2, A3) = 1/e^ ((1/4) + (1/3)) ≡ 0.55

The weight between A1 and A6 is less (signify more closeness) compare to weight between A1 and A3 because the interaction between A1 and A6 is 1/2 is higher than 1/4 between A1 and A3. The weight between A2 and A3 is the lowest among all authors in total papers indicates more closeness.

Using this weight factor on the co-authorship network, we retrieve the most influential author of the query author. All the experiments are performed on the collaborative network using this weight factor. It is discussed in the next section.

Chapter 4

Experimental Evaluation

In this section, we are showing experimental results which we have performed on various datasets on directed and undirected graphs for restricted network. In the graph |V| represents node cardinality, |P| represents data cardinality and D=|P|/|V| represents data density. However, if D=1 then R1NN query returns a point who is situated on the query node. To provide the meaning results, we restricted value of D and performed the experiments for 0.1 to 0.4 data density (D). All the experimental results are showing the average value of 50 queries that we have selected randomly from all the data points. For all the algorithms, we implemented in Java and experimented on an Intel Xenon 2.00 GHz machine in window environment. In our experiment we evaluate the performance of the algorithm on the basis of (i) query processing or experiment cost (time), (ii) number of accessed unique nodes , (iii) number of accessed points, along comparing with a size of requested data point (k), and size of varying data density (D).

4.1 Undirected graph:

For this graph we performed the first experiment on the co-authorship graph of DBLP (http://arnetminer.org/citation) [5]. In co-authorship graph each author signifies the node and is connected by an edge with their co-author if they have published one or more paper together. In previous sections 3.2 we already described co-authorship graph and the proposed weighted scheme between two authors. In this graph every author contains the data point, but for experiments, many of authors, considered as a node and not as data points because of two main reasons. The reasons are: (a) if the author is not active in the community (b) query author, not interested to work with him/her or vise versa. The co-authorship graph on which we have experimented [5] is a connected graph that contains 595775 nodes and more than 2 million edges.

For any given query author q, the RkNN query retrieves all k or less data authors those who has contains query author q in their kNN. Here all the retrieves k or less data authors are those who have more influence towards q as compare to other data authors. Table 1 presents the experimental cost (in Milli Sec or ms) of four previous algorithms [1] and we have verified these results too with the naïve approach, when k varies and D is constant (0.4). The experimental result shows that when k increases, then experimental cost also increases because it explores more nodes and points. While comparing with all algorithms Eager Materialization (Eager_M) performs better and takes less time. Initially Lazy algorithm is taking less time when k =1 compares to Lazy_EP (Lazy Extended Pruning) and naïve but when k increases Lazy takes too much time.

Table 1: Experimental cost (ms) versus k (*D* = 0.4, |*V*| = 595,775)

K	Eager	Lazy	Eager_M	Lazy_EP	Naïve
1	0.50	8.68	0.34	00747.60	2574.94
2	0.72	11514.14	0.54	08537.14	4326.64
4	3.22	69357.84	1.40	11540.68	8295.96
8	10.98	136078.14	9.76	18540.40	16411.06
16	51.14	176051.10	48.92	31153.32	32014.92
32	279.32	200910.70	270.32	62831.12	71649.10

Another experiment has performed when data density D varies and *k* is constant (10). In this setting, experimental outcomes in Table 2 show that when D increases, experiment cost decreases because all the algorithms explores less nodes but more points. Thus, query finds k nearest points early as compare to when D is less and that's the reason the experiment cost is falling.

Table 2: Experimental cost (ms) versus D (*K* = 10, |*V*| = 595,775)

D	Eager	Lazy	Eager_M	Lazy_EP	Naïve
0.1	116.22	154000.74	99.02	21511.42	21261.28
0.2	28.96	152826.67	28.67	18948.82	19189.98
0.3	13.06	128588.02	11.26	16992.28	18973.78
0.4	11.02	115941.02	10.06	16135.48	18720.76

Table 3 and Table 4 shows the result of number of accessing unique nodes and unique points respectively, when D varies and k is constant (10). In Table 3 & 4 Eager and Eager_M accesses approx similar nodes and points, but Eager accesses many nodes, multiple times for different query while performing the range query and takes more time as compare to Eager_M. Similarly, Lazy also performs in the same way and takes too much time because of its point centric approach.

Table 3: Accessed nodes versus D (*K* = 10, |*V*| = 595,775)

D	Eager	Lazy	Eager_M	Lazy_EP	Naïve
0.1	4404	181979	4368	197452	305513
0.2	1262	147058	1231	164310	303763
0.3	457	112706	453	129400	284643
0.4	335	90787	330	106886	255844

Table 4: Accessed points versus D (*K* = 10, |*V*| = 595,775)

D	Eager	Lazy	Eager_M	Lazy_EP	Naïve
0.1	496	19958	483	21728	33768
0.2	319	36734	306	41171	75926
0.3	220	48360	210	55533	122226
0.4	196	60564	195	71198	170412

Other experiments, we performed for the graph on the road network. In road network datasets, we accessed from this challenge (http://www.dis.uniroma1.it/challenge9/download.shtml). In this USA road network weighted datasets, we worked for NW (north-west) USA road segment distance graph that contains 1.2 million nodes and 2.8 million arcs or edges. Here the nodes signify the intersections between the roads and the weight on the edges shows the distance between two intersections. In interesting query here for intersecting query point (road) is checking their top most affected intersecting point. Table 5 shows the execution cost when k varies and D constant. The experimental result in table 5 also shows that the similar scenario when k is increasing then the execution cost is also increasing.

Table 5: Experimental cost (ms) versus k (D = 0.4, $|V|$ = 595,775)

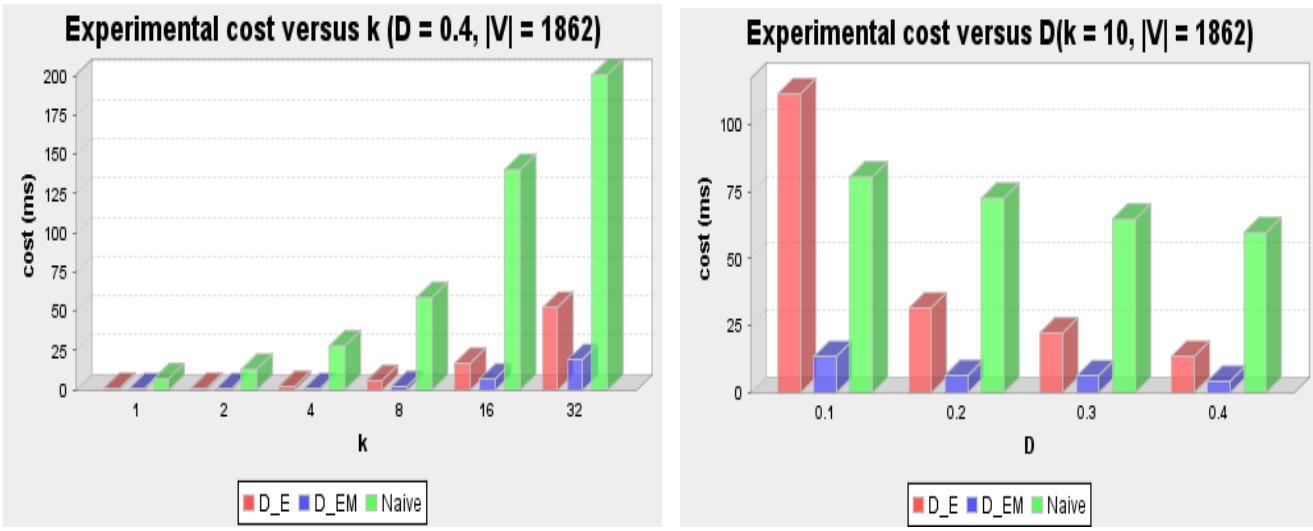
K	Eager	Lazy	Eager_M	Lazy_EP	Naïve
1	0.50	8.68	0.34	747.60	2574.94
2	0.72	11514.14	0.54	08537.14	4326.64
4	3.22	69357.84	1.40	11540.68	8295.96
8	10.98	136078.14	9.76	18540.40	16411.06
16	51.14	176051.10	48.92	31153.32	32014.92
32	279.32	200910.70	270.32	62831.12	71649.10

4.2 Directed graph:

We performed all the experiments for this graph on **naïve** algorithm, directed eager (**D_E**) & directed eager materialization (**D_EM**) algorithm.

The first set of experiments has been performed for this graph on a social network dataset of Facebook (http://toreopsahl.com/datasets/#online_social_network) [7]. The dataset contains 1862 users and 20k directed ties among the users. User is represented as a node 'n' and directed edge between two users' e(i,j) if user 'i' sent at least one online message to user 'j'. The strength measure or weight on the edge e (i,j) represents the 1/(number of messages) between users 'i' to 'j', signify more messages than less weight and higher the closeness between the users. For retrieving efficient query, the top most influence neighbors of the user requires a strong strength measure or a weight that defines the 'nearness' (closeness) between the users. According to the paper [4] for measuring the user influence we require various parameters in consideration which differ from social network to network. For deciding a good strength measure, some other social networking parameter such as comments, likes, mentions, share and others are necessary. But lack of complete datasets, we were unable to do that and left it for the future work.

In the first experiment, we evaluate the experimental cost (in milli sec. Or ms) when k increases and D is constant (0.4). Figure 4.1 (a) shows the result, when k increases the cost also increases because it accesses the more nodes and more points for retrieving the more number of k and takes more time for pruning the search space. In a second experiment, it is illustrated that when D increases and k constant (10), then experimental cost decreases because it finds more point around the query and least number of nodes. In figure 4.1 (b) experimental result shows that when data density D is low, D_E performs worst comparisons to naïve approach because D_E accessed more nodes for retrieving the k point, but when D increases cost of D_E drastically decreases and performs better compared to the naïve approach. During this experiment, we observe that when D increases, then the number of accesses nodes decreases and the number of points increases.



(a) (b)

Figure 4.1: Directed Graph of Facebook social network, Experimental cost (ms) versus- (a) *k* (requested points) (b) *D* (data density)

The second set of experiments has been performed for this graph on product co-purchasing network of Amazon (http://snap.stanford.edu/data/amazon0505.html) [6]. The datasets contain 410,235 nodes and 3.3 million edges. Each node represents the product which was purchased by any customer from the Amazon sites. Two products (like i & j) are linked by a directed edge from i to j if a product i commonly co-purchased with product j. All the edge weights are assigned randomly just to experiment, but it is possible to assign some effective weight for effective query and that requires the lot of co-purchasing factor to consider that. That we left for future work.

Table 6: Experimental cost (ms) versus k (*D* = 0.4, |*V*| = 410,235)

K	D_E	D_EM	Naïve
1	0.56	0.32	1424.08
2	0.64	0.36	2460.92
4	1.18	0.52	5138.54
8	3.36	1.62	11371.42
16	9.94	4.28	24444.34
32	37.68	13.74	48922.28

Table 6 analyzed the experimental cost when k varies and D is constant. The experimental result shows that when k increases, then experimental cost also increases. In evaluation naïve algorithm takes much time compares to D_E because D_E prune the search space early and explored only a few nodes. While naïve approach had been explored all the points and checked if that contain the query point or not in within a required set of points. In figure 4.2, the overall best performance shown by the directed_eager materialized (D_EM) algorithm compares with the directed eager (D_E) because it accessed the materialized information instead of applying Range and Verify queries. (Naïve showed worst amongst all and not shown in the graph)

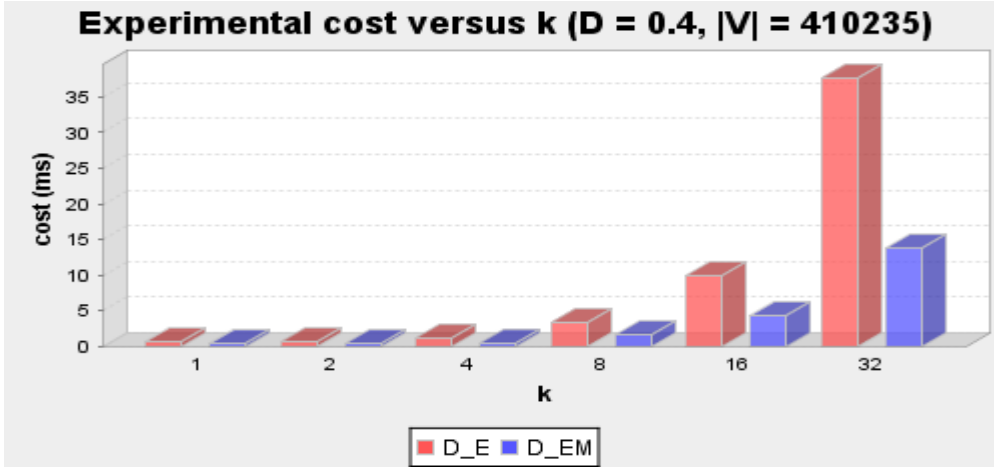
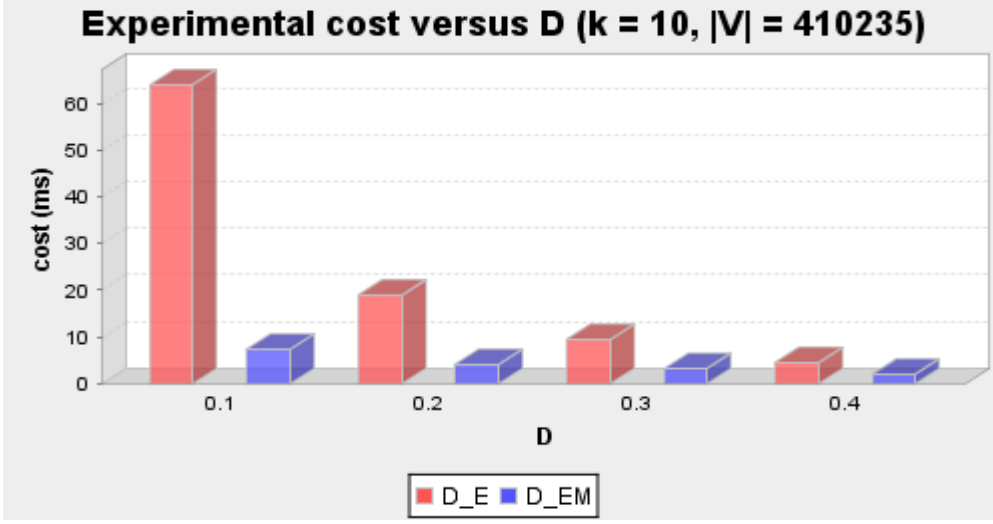


Figure 4.2: Directed Graph of Amazon product co-purchasing network

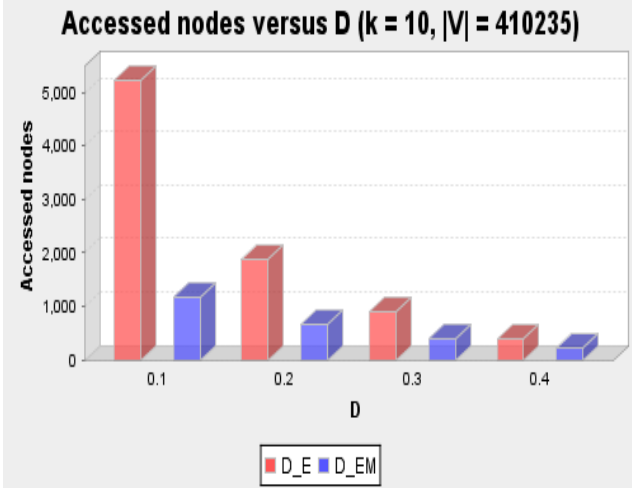
Another experiment we performed for the experimental cost when D varies and k is constant. Table 7 illustrates that when D increases then experimental cost proportionally decreases because the algorithm finds more points near around and expands very few nodes. Figure 4.3(a) shows this experimental for D_E and D_EM algorithms. Figure 4.3(b) & 4.3(c) are showing accessed unique nodes and unique point respectively when D varies and k constant.

Table 7: Experimental cost (ms) versus D (k = 10, $|V|$ = 410,235)

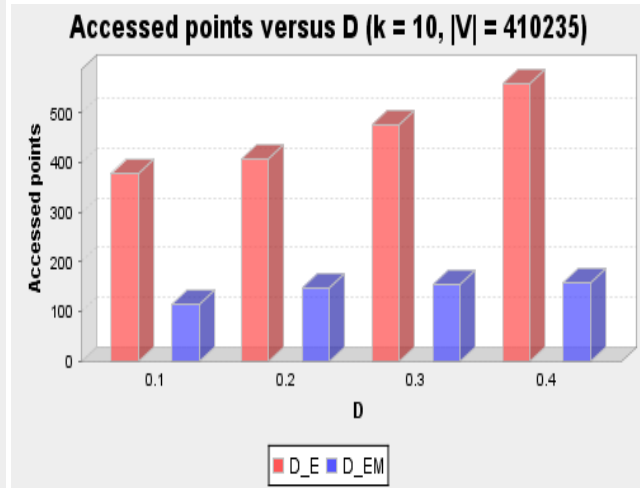
k	D_E	D_EM	Naïve
1	64.06	7.32	16852.38
2	19.1	4.2	15140.12
4	9.56	3.2	14445.6
8	4.58	2.08	13986.24



(a)



(b)

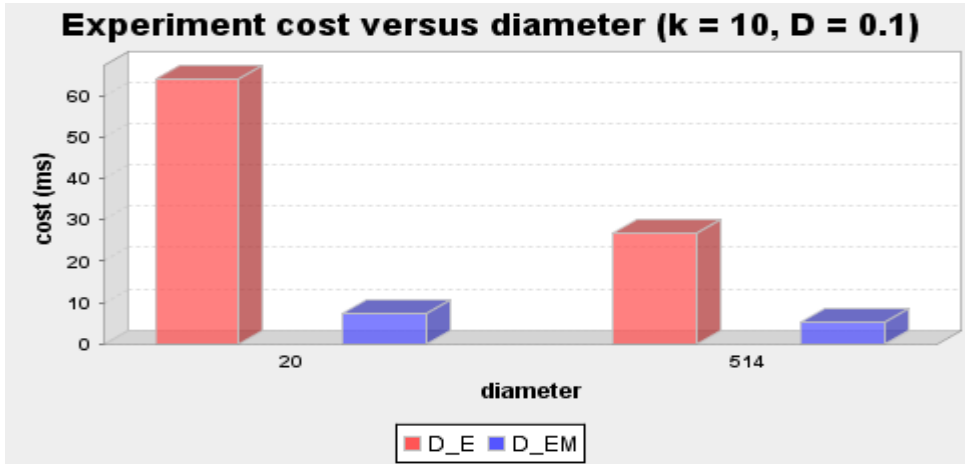


(c)

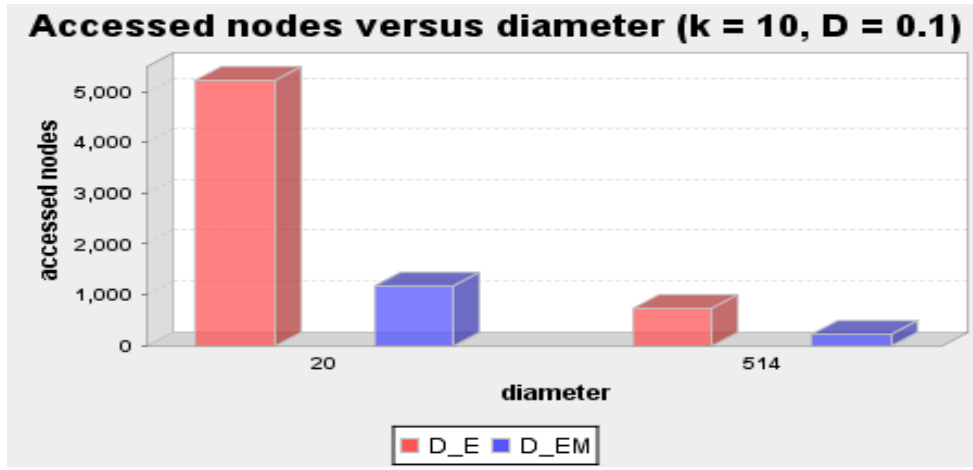
Figure 4.3: Directed graph of Amazon product co-purchasing network, data density D versus- (a) Experimental cost (ms), (b) Accessed nodes, and (c) Accessed points

Next experiment, we performed on to direct web graph where nodes represent web pages and edges represents hyperlinks between them (https://snap.stanford.edu/data/index.html#web) [8]. The first directed web graph is Berkeley-Stanford web dataset that contains 685,230 nodes and 7.6 million edges. Nodes represent the web pages of berkeley.edu and stanford.edu domains while the edges show the hyperlinks between them. The diameter d (longest shortest path) of this network is 514. For the experiments, edge weight assigned randomly.

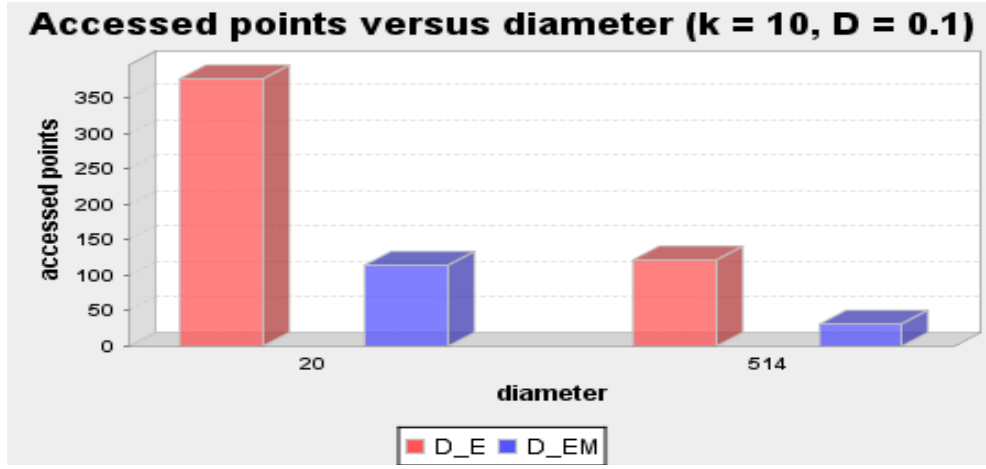
In this experiment, we evaluate the cost (ms) on two datasets when the diameter varies. The two data sets, first is Amazon co-purchasing graph and second is Berkeley-Stanford web graph, whose diameter respectively is 20 and 514. In figure 4.4, experiment shows that when diameter is less than the experimental cost is more comparable when the diameter is high because the query expands most of the network by accessing number of more, nodes and points within the range. In large networks like Amazon co-purchasing network is contain 410,236 products (nodes) and diameter d between them is 20 means higher connectivity between products and that's the reason it accesses most of the nodes and points during the query processing.



(a)



(b)



(c)

Figure 4.4: Directed graph of Amazon product co-purchasing network (diameter 20) and Berkeley-Stanford web graph (diameter 514), diameter versus- (a) Experimental cost (ms), (b) Accessed nodes, and (c) Accessed points

Chapter 5

Conclusion and Future work

In this work, we experimented on a few applications of undirected graph and for that we used R*k*NN algorithms that were already published in paper [1]. We also checked the proof of correctness of this algorithm with the naïve approach. The most important application of an undirected graph, we performed for the co-authorship graph of DBLP. In this dissertation, we have provided better 'closeness (weight) factor' between authors and retrieve the top-k influential co-author of a query author and analyzed through various experiments.

The algorithms have been proposed in this work for the directed graph. This thesis concludes the two algorithms for directed network along with optimized version and has also checked the correctness of algorithm through the naïve approach. The D_EM algorithm performs better amongst all and takes very few times. We performed the set of experiments for various applications on directed and undirected network. In our experiment we cover application including from social networks, web graph, product Co-purchasing network, for directed graph and co-authorship graph, road network for undirected graphs. Future work concerns in direction of some better graph storage technique.

Bibliography:

[1] Yiu, Man Lung, Dimitris Papadias, Nikos Mamoulis, and Yufei Tao. "Reverse nearest neighbors in large graphs." *Knowledge and Data Engineering, IEEE Transactions on* 18, no. 4 (2006): 540-553.

[2] Newman, Mark EJ. "Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality." *Physical review E* 64, no. 1 (2001): 016132.

[3] Bukvova, Helena. "Studying research collaboration: a literature review." (2010).

[4] Cha, Meeyoung, Hamed Haddadi, Fabricio Benevenuto, and P. Krishna Gummadi. "Measuring User Influence in Twitter: The Million Follower Fallacy." *ICWSM* 10 (2010): 10-17.

[5] Tang, Jie, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. "Arnetminer: extraction and mining of academic social networks." In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 990-998. ACM, 2008.

[6] Leskovec, Jure, Lada A. Adamic, and Bernardo A. Huberman. "The dynamics of viral marketing." *ACM Transactions on the Web (TWEB)* 1, no. 1 (2007): 5.

[7] Opsahl, Tore, and Pietro Panzarasa. "Clustering in weighted networks." *Social networks* 31, no. 2 (2009): 155-163.

[8] Leskovec, Jure, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters." *Internet Mathematics* 6, no. 1 (2009): 29-123.

[9] Korn, Flip, and S. Muthukrishnan. "Influence sets based on reverse nearest neighbor queries." In *ACM SIGMOD Record*, vol. 29, no. 2, pp. 201-212. ACM, 2000.

[10] Stanoi, Ioana, Divyakant Agrawal, and Amr El Abbadi. "Reverse Nearest Neighbor Queries for Dynamic Databases." *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. 2000.

[11] Safar, Maytham, Dariush Ibrahimi, and David Taniar. "Voronoi-based reverse nearest neighbor query processing on spatial networks." *Multimedia systems* 15, no. 5 (2009): 295-308.

[12] Tran, Quoc Thai, David Taniar, and Maytham Safar. "Reverse k nearest neighbor and reverse farthest neighbor search on spatial networks." *Transactions on large-scale data-and knowledge-centered systems I*. Springer Berlin Heidelberg, 2009. 353-372.

[13] Cheema, Muhammad Aamir, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Xuefei Li. "Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks." *The VLDB Journal—The International Journal on Very Large Data Bases* 21, no. 1 (2012): 69-95.