# Finding RkNN Set in Directed Graphs

Pankaj Sahu[1], Prachi Agrawal[2], Vikram Goyal[1], and Debajyoti Bera[1]

[1] Indraprastha Institute of Information Technology-Delhi (IIIT-D), India
{pankaj1244,vikram,dbera}@iiitd.ac.in
[2] LNM Institute of Information Technology, Jaipur, India
happyprachi.1@gmail.com

**Abstract.** The *reverse $k$-nearest neighbors* of a query data point $q$ characterizes the influence set of $q$, and comprises of data points which consider $q$ among their $k$-nearest neighbours. This query has gained considerable attention due to its importance in various applications involving decision support systems, profile-based marketing, location based services, etc. Although this query is reasonably well-studied for scenarios where data points belong to Euclidean spaces, there has not been much work done for non-Euclidean data points, and specifically, for large data sets with arbitrary distance measures. In this work, a framework has been proposed for performing RkNN query over data sets that can be represented as directed graphs. We present a graph pruning technique to compute the RkNN of a query point which significantly reduces the search space. We report results of extensive experiments over some real-world data sets from a social network, a product co-purchasing network of Amazon, the web graph, and study the performance of our proposed heuristic in various settings on these data sets. These experiments demonstrate the effectiveness of our proposed technique.

## 1 Introduction

A common problem that arises in many marketing and decision support systems is to determine the "influence" of a data point on other data points of a database. Korn, Flip and Muthukrishnan introduced the concept of *reverse nearest neighbor*[4] in 2000 to compute the set of influenced points for datasets with a notion of "closeness" between the points. The underlying idea is that a nearby point have a larger influence than a point farther away, which immediately leads to the concept of reverse nearest neighbor (RNN), and its generalisation reverse k-nearest neighbor (RkNN).

The most common application of reverse nearest neighbor query for computation of influential sets can be illustrated by a facility location scenario. Suppose a company is exploring the option of opening a new restaurant in a location and wants to find likely customers – people who would be "more likely" to use this restaurant compared to another one; in our terminology, we say these people are influenced by the location and profile of this restaurant.

The important issue, therefore is to efficiently compute the data points influenced by a query point. Technically, the reverse nearest neighbor (RNN) of a query point consists of all those data points for which the query point is their nearest neighbor (NN). Reverse k-nearest neighbor (RkNN) is a generalisation of RNN, where kNN is used in the place of NN in the earlier definition.

R. Natarajan et al. (Eds.): ICDCIT 2015, LNCS 8956, pp. 162–173, 2015.

Efficient algorithms for computing RkNN have been designed for various types of data [2,3] and several variants of nearest neighborhood query. Majority of these works have focused on data with a well-defined concept of distance or similarity, such as Euclidean distances, similarity measures for word vectors, and other distances that are metric in nature. However, the problem becomes fundamentally different for data points with arbitrary distances between the points such as people, products, movies, etc. Data with an arbitrary notion of distance is well represented by a weighted graph. The focus of this paper are data which can be represented as such graph, and furthermore, as weighted directed graphs. The nodes in the graph are the data points, and the weighted and directed edges represent the distance and relation, respectively, between related points.

Given such a graph, it is often meaningful to consider the shortest-path distance as the distance between two points. There are a few major hurdles in computing RkNN of a query point in such a graph. First similar to the general problem, the concept of nearest neighborhood is not symmetric, so we cannot simply run one single-source shortest path query from the query point.

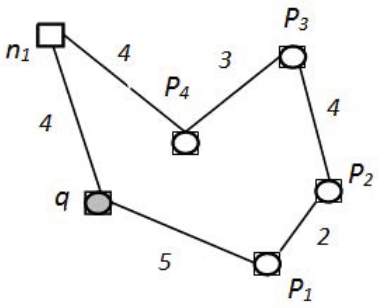



**Fig. 1.** Nearest Neighborhood using Shortest-path Distance

For example, in Fig. 1, $P_1$ is the nearest neighbor of query point $q$ but $P_2$ is the NN of $P_1$ and not of $q$. This also implies that the points in RNN need not be in the immediate vicinity of the query point, and a large segment of the graph may have to be traversed. This presents a major challenge for large graphs because we cannot benefit from locality conditions. A solution based on running a Dijkstra's algorithm from every point (to compute the kNN of each point) would be correct, but computationally very expensive. Secondly, we are interested in a non-index based solution; index/pre-processing based techniques are often not much efficient or directly possible for situations where $k$ is not fixed and due to relationship constraints present in graphs.

An interesting application of RkNN query in directed graphs can be seen in social networks; here, RkNN can be used to determine the influence set of any person. The weights on the directed edges of such network can represent the influence of one person on another, say, regarding forwarding of messages, and the RkNN query for a person would then give a set of individuals who would play an important role in diffusing any information initiated or forwarded by the query person.

In an earlier work, a graph-pruning based algorithm was designed by Papadias et al.[11] for undirected graphs. However their technique does not work for directed graphs. The main contribution in this paper is an approach for computing reverse nearest neighborhood for directed graphs. We give two algorithms, Directed Eager (D_M) and an optimized version for large graphs, Directed Eager Materialization (D_EM) for processing (monochromatic) RkNN for arbitrary $k$. Another important contribution is extensive experimentation on large directed networks coming from real datasets including that of a social network, a product co-purchasing network from Amazon and a web network of Berkeley-Stanford university.

The rest of the paper is organized as follows: Section 2 presents the necessary background and related work. Section 3 discusses our algorithm. Section 4 reports our experimental results on some real world data sets including performance comparisons with respect to various parameters. Section 5 concludes the paper suggesting some directions for future work.

## 2 Background and Related Work

### 2.1 Reverse Nearest Neighbor Query

We will now formally define the relevant terms.

Given a dataset $P$, the monochromatic RkNN of a query point $q$ not in $P$ is defined as:

$$\textbf{RkNN (q)} = \{\mathbf{p} \in \boldsymbol{P} \mid \textbf{dist(p, q)} \leq \textbf{dist(p, p}_k\textbf{(p))}, \textbf{ where p}_k \in \boldsymbol{P} \textbf{ is the k}^{th} \textbf{ NN of p}\}$$

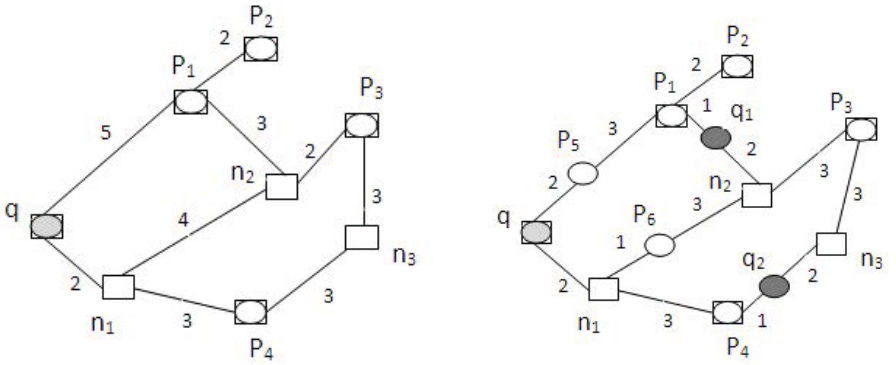
Similarly, given two datasets $P$ and $Q$ and a query point q$\in Q$, the bi-chromatic RkNN of a query point q returns all those data points $p \in P$ which are nearer to $q$ than any other points of $Q$.

$$\textbf{bRkNN (q)} = \{\mathbf{p} \in \boldsymbol{P} \mid \textbf{dist(p, q)} \leq \textbf{dist(p, q}_k\textbf{(p))}, \textbf{ where q}_k \in \boldsymbol{Q} \textbf{ is the k}^{th} \textbf{ NN of p}\}$$

In this paper, we are interested in only monochromatic RkNN.

Furthermore, like [11], we consider a generalized scenario where not all nodes in the graph are data points. For instance, not every author in a co-authorship graph works in some specified field. Such networks are called *restricted* networks. Consider the example in Fig. 2(a). Suppose it represents a DBLP co-authorship (collaboration) network. An (monochromatic) R1NN query for query point $q$ will return authors whose 1NN is $q$, among authors of his same field $\{P_1, P_2, P_3, P_4\}$. Other nodes, such as $n_1$, $n_2$ and $n_3$ represent the authors which are not working in the author's field and, thus, are irrelevant for this query.

In this figure, RNN($q$)=$\{P_4\}$, because $P_4$'s NN is $q$, and R2NN($q$)=$\{P_1,P_2,P_4\}$. The other type of network, where the data and query points can be any node of the graph, is called an unrestricted network. However, we give a solution for the more general restricted network.




(a) Monochromatic RNN query (b) Bi-chromatic RNN query

**Fig. 2.** Types of RNN Queries

### 2.2 Related Work

A lot of work has been done on RNN queries. The work was started by a paper [4] in 2000 where the authors define the notion of influence set and show that nearest neighbour set of a point is different from a reverse nearest neighbour set of the point. They show multiple applications of RNN query whereas finding a facility location is one of them. To solve RNN query efficiently, the authors propose to have a pair of R-tree structures on spatial objects and MBR regions of objects containing their nearest neighbour, respectively. Subsequently, there has been more work [7,10] on providing efficient solutions using better data structures or methods constraining the search to a well defined set of candidates. Our work focuses on finding RkNN set for weighted directed graphs and uses network distances as well relationship of nodes for search.

The other work for RNN query has been done on Road networks [8,5,11,9,6] and for continuous RNN queries [1,3]. In continuous RNN queries case, the problem is to continuously report the RkNN set of a moving query point. The brute force solution of computing RNN set afresh would not be good and hence an incremental solution was proposed by the authors. The approach is based on the concept of safe region that defines the boundary around a query point, crossing which there is a need to recompute the RNN set.

The work on RNN query over road network was done in [8,5]. Safar et al. [5] used a network Voronoi diagram (NVD) for efficiently processing the RNN queries over road 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. The authors in [11] model the road network as an undirected graph and presents two versions of problem, restricted case where data objects/points can be only at intersection nodes and unrestricted case where objects can be anywhere on the edges. The authors gave two different algorithms called eager and lazy approaches that use heuristic rules to prune the search space.

## 3 Proposed Solution

Here we present our proposed algorithms for solving an RkNN query for a given directed graph (digraph) G, and edge weight function $d(\cdot)$. As explained earlier, our algorithm is based on the results of Papadias et al.[11] with crucial modifications necessary to handle directed networks.

We maintain two graphs, one $G_M$ and another graph $G_R$ whose edges are reverse of that of G. This is required because our algorithm traverses in a best-first manner starting at $q$ (which uses $G_M$), but to ascertain the inclusion of a node p in RkNN($q$) we need to check the neighborhood of $p$ for which we use $G_R$.

Our algorithm is based on the following key lemma which is used to optimize exploration of the graph. Using the rule given in the lemma, unpromising nodes are identified those need not be further explored. As exploring them further would not be useful due to the reason that if a successor of the node have its short path to the query point through the node, it would not be in RkNN.

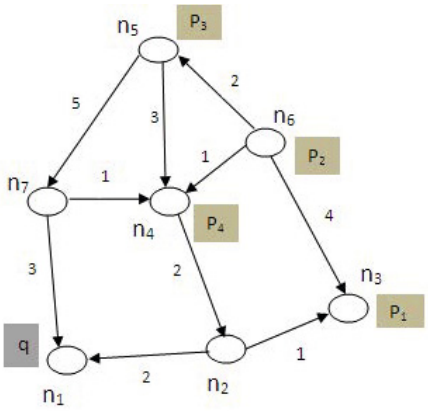



**Fig. 3.** Lemma

**Lemma 1.** *For a query point q, a data point p and a node n, if* $d(n, q) > d(n, p)$ *and there is another data point p′≠ p whose shortest path to q passes through n, then point p′ is not in the RNN set of q.*

*Proof.* The proof is really straight forward.
*d(p′,q) = d(p′,n)+d(n,q) >d (p′,n)+d(n,p) ≥ d(p′,p).*
Since, d(p′,p)<d(p′,q), p′ ∉ RNN(q). □

Therefore, in such a situation, the node $n$ can be pruned and need not be further expanded. As an illustration, refer to Fig. 3. There, $d(n_2, q) = 2 > d(n_2, p1) = 1$. According the lemma, any data point (in Fig. 3 $p_4$) whose shortest path to $q$ passes through $n_2$ cant 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 need not be expanded further.

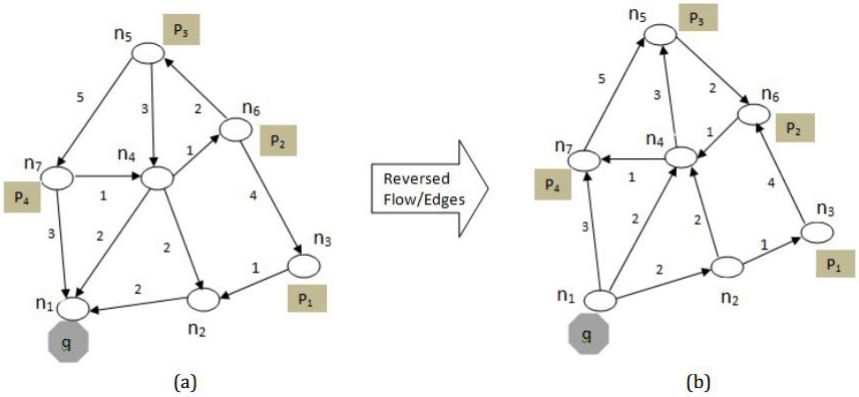



**Fig. 4.** Example of Directed Eager (a) Main Graph $G_M$ (b) Reverse Graph $G_R$

Similar to [11], we use two subroutines *range-NN* query and *verify* query. A straight forward modification of these subroutines given in their paper turn out to be sufficient for directed graphs, so we skip their implementation details in our paper. The *range-NN(n,k,q)* query returns (at most) $k$ nearest data points with shortest-path distance smaller than $d(n, q)$. The query *verify(n,k,q)* is similar to *range-NN(n,k,d(n,q))*, except that it stops as soon as q is encountered.

### 3.1 Directed Eager(D_E) Algorithm

Our main algorithm traverses the reverse graph $G_R$ starting from the query point q, in a best-first manner, and every encountered node n in $G_R$ is inserted into a min-heap H. H is also used to select the next node to explore. For every point node n deheaped from the queue, we proceed as given below. The range-NN and verify queries are executed on the main digraph $G_M$ and all other operations are done on the reverse graph $G_R$. The only difference between the range-NN query and the verify query is that range-NN returns the set of k nodes whereas verify returns boolean value to decide whether to explore the node further.

- If the node contains a data point p∈P, then we need to take two decisions:
  1. Whether $p \in$ RkNN($q$)? This we determine by running *range-NN(p,k,q)*.
  2. Whether p should be explored further? This we determine using Lemma 1 by using *verify(p,k,q)*. If $k$ (or more) points are found, then there is no need to explore further the neighbors of $p$.
- If the node does not contain a data point, then we only need to determine whether that node should be explored further. We employ the same treatment as (2) above.
- In the case of node is not to be explored further, no adjacent nodes of the currently explored node with respect to $G_R$ are enqueued in the heap $H$.

We now illustrate our algorithm on Fig. 4, on which we run a query RNN(q) for k=1. The algorithm starts its traversal from node $n_1$ (source node which contains the query point q) and insert $<$ $n_1$(q),0$>$ into H (currently empty). After this, $n_1$ is deheaped.

**Algorithm. D_Eager**($q$, $k$)
insert $< n(q), 0 >$ into $H$ // $n(q)$ is the node containing query point $q$
**while** *(not-empty(H)* **do**
  $< n, d(q, n) >$:= de-Heap($H$);
  **if** *($n$ is not visited before)* **then**
    mark $n$ as visited
    **if** *($n$ does not contain a point)* **then**
      $kNN(n)$ = *range-NN(n,k,d(q,n))* in main digraph $G_M$.
    **end**
    **else if** *($n$ contains a point $p$)* **then**
      *kNN(p)* = *verify(p,k,q)* too check in main digraph $G_M$.
      if $q$ discovered by verification add $p$ to *RkNN(q)*
    **end**
    **if** *(*$| kNN(n) | < k$ *or* $| kNN(p) | < k$*)* **then**
      For (each non-visited adjacent node $n_i$ of $n$ in reverse digraph $G_R$)
      insert $< n_i, d(q, n) + w(n, n_i) <$ into $H$
    **end**
  **end**
**end**
return RkNN($q$)

Since it does not contain any data point, and since d($n_1$,q)=0, range-NN returns the empty set trivially. Therefore, it's adjacent nodes in reverse graph $G_R$, <$n_2$, 2> <$n_4$, 2> and <$n_7$, 3>, are now inserted into $H$.

The next deheaped node is $n_2$ which also does not contain a data point. A range-NN($n_2$,1,2) query in $G_M$ again does not return any point within distance 2 from $n_2$. Similar to the previous case, its adjacent node $< n_3$, 3> in $G_R$ is inserted into $H$.

The subsequent deheaped node is $n_4$, which too, does not contain a data point. However, a range-NN($n_4$,1,2) in $G_M$ returns a data point $p_2$, because $d(n_4, p_2) = 1 < d(q, n_4) = 2$. Hence node $n_4$ can be pruned, and so its adjacent nodes are not inserted into $H$.

Next node $n_7$ is deheaped which contains a data point $p_4$. We run a *range-NN($n_7$,1,3)* query that returns $p_4$ itself, however running *verify($p_4$,1,q)* in $G_M$ returns the data point $p_2$. Therefore, neither $p_4$ belongs to RkNN($q$), nor should $p_4$ be explored further. In case of verify query at a node, data point located at the node itself is not considered.

At last node $n_3$, containing data point $p_1$, is deheaped and *range-NN($n_3$,1,3)* returns point $p_1$. The *verify($p_1$,1,q)* query in $G_M$) finds $q$ and returns true. So $p_1$ is added into RkNN($q$) and no adjacent of $n_3$ is added in heap $H$.

At this stage, $H$ is empty which means that our algorithm is terminated with final result containing RNN($q$)=$\{p_1\}$.

### 3.2 Directed Eager Materialization

Materialization is the technique of pre-computing the shortest distances between all pairs of nodes and store it in a lookup table. A naïve approach for materialization is to apply the kNN query on each node, but this method is not scalable for large graph.

Therefore, an *all-NN* algorithm is proposed that accessed the network only once to compute the distance value for different values of $k$ for each node.

The algorithm is as given below.

**Algorithm. Directed_All-NN** ($k$)
**for** *(each node n those contains point p in main digraph* $G_M$*)* **do**
  insert $< n, p, 0 >$ into $H$
**end**
**while** *(not-empty(*$H$*))* **do**
  $< n, p, d(p, n) >$:= de-Heap($H$)
  **if** *(* $\mid kNN(n) \mid < k$ *and* $p \notin$ *kNN(n) )* **then**
    add $< p, d(p, n) >$ to kNN($n$)
    **for** *(each adjacent node* $n_i$ *of* $n$ *in reverse graph* $G_R$*)* **do**
      **if** *(* $\mid kNN(n_i) \mid < k$ *and* $p \notin$ *kNN(*$n_i$*) )* **then**
        insert $< n_i, p, d(p, n)$+$w(n, n_i) >$ into $H$
      **end**
    **end**
  **end**
**end**
return kNN

Our directed eager materialization algorithm (Directed_EM or D_EM) utilizes the materialized lookup table build above to retrieve the kNN of any n in constant time. This replaces the costly range-NN and verify operations to constant time table look-up operation.

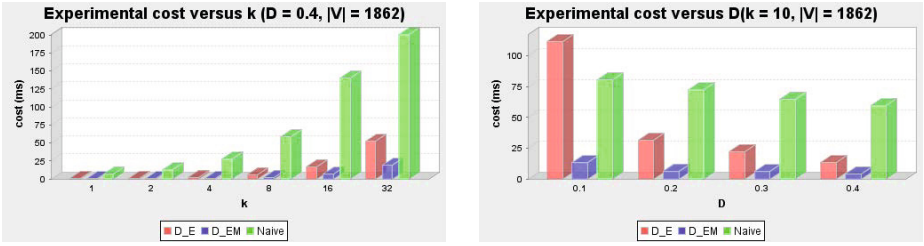
## 4 Experimental Evaluation

In this section, we show our experimental results for processing RkNN query over directed graphs. In the graph |V| represents number of nodes in the graph, |P| represents data points cardinality and D=|P| / |V| represents data density. However, if $D = 1$ then R1NN query returns a point situated on the query node itself. For providing more meaningful results, we restricted value of D from 0.1 to 0.4. All the experimental results show the average value of 50 queries generated randomly from all the data points in the network. All algorithms are implemented in Java and experiments are run on an Intel Xenon 2.00 GHz machine with Windows environment. In our experiments, we evaluate the performance of the algorithms in terms of (i) computation time in milliseconds(ms), (ii) number of accessed nodes, (iii) number of accessed points. We study the performance with respect to the size of requested data points ($k$), and the density ($D$) of a graph. The value of $k << \mid P \mid$.

### 4.1 Directed Graph

We study the performance of directed eager (D_E), directed eager materialization (D_EM) and the naïve approach on different datasets.

The first set of experiments has been performed on a social network dataset of Facebook [1]. The dataset contains 1862 users and 20k directed edges among the users. A user is represented as a node n and an edge direction between two users e(i,j) refers, if a user 'i' has sent at least one online message to a user 'j'. The strength measure or weight on the edge e(i,j) represents the 1/(number of messages) between users 'i' to 'j'.




(a) k (requested points) (b) D (data density)

**Fig. 5.** Facebook Social Network (a) Effect of $K$ on Computation Time (ms) (b) Effect of $D$ (data density) on Computation Time

In the first experiment, we measure the effect of query parameter $k$ on computation time when density $D$ is 0.4. Figure 5(a) shows the result. When $k$ increases, the computation cost also increases. It is because of the increase in the number of accesses of nodes/points with increase in $k$. In the second experiment, effect of density $D$ on the computation cost is evaluated. Fig. 5(b) shows the results. It is seen that when $D$ increases (value of $K$ is set to 10 as a default value), the computation time cost decreases. It is due to the fact that increase in density of points in the graph makes availability of data points near to the query points as well as increases the proximity of data points to each other and other graph nodes. The probability of finding k data points within the distance d(n,q) from a node increases and hence the adjacent nodes of the node need not be put in the heap. This results in better pruning and less graph is searched for RkNN.

It can be seen that when the data density $D$ is very low, $D\_E$ performs worse as compared to the naïve approach. It is because of non-pruning of nodes. Most of the nodes in the graph do not find k data points and hence their adjacent nodes are enqueued in the heap, and the algorithm $D\_E$ ends up in accessing more nodes for retrieving $k$ points. When $D$ increases, the computation cost of algorithm $D\_E$ drastically decreases and performs better than the naïve approach. During this experiment, we also observe that when $D$ increases, then the number of accesses nodes decreases and the number of points increases, because more points need to be verified.

The second set of experiment has been performed on product co-purchasing network of Amazon [2]. The dataset contain 410,235 nodes and 3.3 million edges. Each node represents the product which was purchased by a customer from the Amazon sites. Two products $i$ and $j$ are linked by a directed edge from $i$ to $j$ if the product $j$ is

[1] (http://toreopsahl.com/datasets/#online_social_network)
[2] http://snap.stanford.edu/data/amazon0505.html

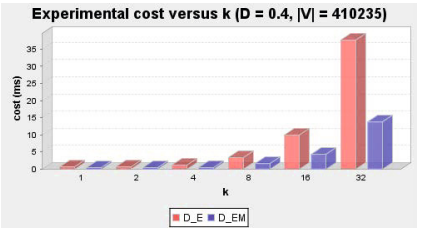
purchased after product $i$. The edge can be interpreted as a causal relation, i.e., if item $i$ is purchased then item $j$ is also purchased. All the edge weights are assigned randomly for our experiments.

**Table 1.** Effect of $K$ on Computation Time, $D = 0.4$, Amazon Dataset

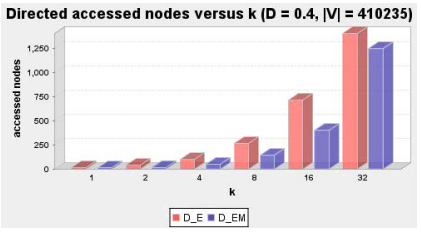
| K | D_E | D_EM | Naive |
|---|---|---|---|
| 1 | 0.56 | 0.32 | 1424.08 |
| 2 | 0.64 | 0.36 | 2460.92 |
| 4 | 1.18 | 0.52 | 5138.54 |
| 8 | 3.36 | 1.62 | 11371.42 |
| 16 | 9.94 | 4.28 | 24444.34 |
| 32 | 37.68 | 13.74 | 48922.28 |

**Table 2.** Effect of $D$ on Computation Time, $K = 10$, Amazon Dataset

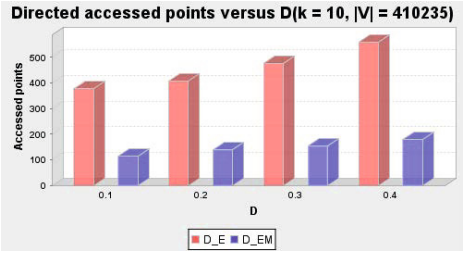
| D | D_E | D_EM | Naive |
|---|---|---|---|
| 0.1 | 64.06 | 7.32 | 16852.38 |
| 0.2 | 19.1 | 4.2 | 15140.12 |
| 0.3 | 9.56 | 3.2 | 14445.6 |
| 0.4 | 4.58 | 2.08 | 13986.24 |




(a)




(b)



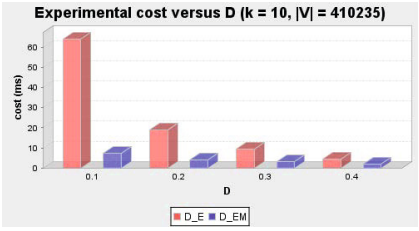

(c)

**Fig. 6.** Amazon product co-purchasing Directed network (a) Effect of $k$ on Computation Time (b) Effect of $k$ on Accessed Nodes (c) Effect of $k$ on accessed Points

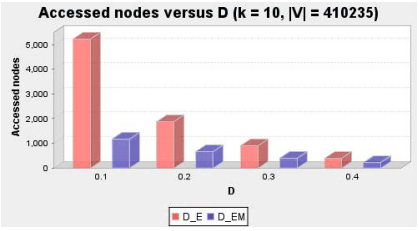
Table 1 shows the computation cost for different values of $k$ when $D$ is 0.4. The experimental results show that when $k$ increases, then computation time increases. The naïve algorithm takes too much time as compared to $D\_E$. It is because of early pruning of the search space by algorithm $D\_E$ and exploring only few nodes. Fig. 6 shows the performance of the directed_eager materialized (D_EM) algorithm with respect to the directed eager(D_E) algorithm. (D_EM) algorithm performs much better due to its

constant time cost for verification step (Naïve showed worst amongst all and not shown in the graph). However the graph for the value of $D$ as 1 are not shown, in this case $D\_E$ would perform best as the probability of finding $k$ data points is very high within $k$ hops from the node. As we will go farther from the query point, their probability of being in RkNN set would decrease.

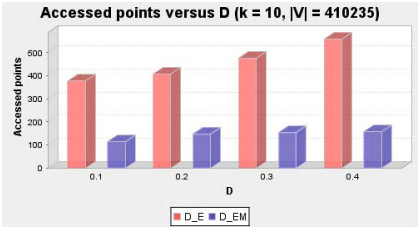
Next experiment shows the effect of $D$, given in Table 2. It shows that when $D$ increases the computation cost decreases. It is because of the reason that the algorithm $D\_E$ finds more points near around the query point and expansion happens to a small set of nearby nodes. Figure 7 shows this experimental result for $D\_E$ and $D\_EM$ algorithms. Figure 7(b) & 7(c) show qualitative results in terms of accessed nodes and accessed points. It may be noted that we do not count an accessed node twice.




(a)




(b)




(c)

**Fig. 7.** Amazon product co-purchasing Directed network (a) Effect of $D$ on Computation Time (b) Effect of $D$ on Accessed Nodes (c) Effect of $D$ on accessed Points

## 5 Conclusion and Future Work

We have presented two algorithms for RkNN query on directed graphs in the paper. The optimized version, $D\_EM$ algorithm, performs best amongst all and takes very less time for finding out an RkNN set. We have performed an extensive set of experiments on three real graphs to study the performance of the algorithms. Results show that the pruning rule we have proposed in the paper is very effective.

**Acknowledgement.** Authors will like to acknowledge the support provided by ITRA project, funded by DEITy, Government of India, under grant with Ref. No. ITRA/15(57)/ Mobile/HumanSense/01

## References


1. Cheema, M., Zhang, W., Lin, X., Zhang, Y., Li, X.: Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. The VLDB Journal 21(1), 69–95 (2012)
2. Goyal, V., Likhyani, A., Bansal, N., Liu, L.: Efficient trajectory cover search for moving object trajectories. In: Proceedings of the 2013 IEEE Second International Conference on Mobile Services, MS 2013, pp. 31–38. IEEE Computer Society, Washington, DC (2013)
3. Goyal, V., Navathe, S.B.: A ranking measure for top-k moving object trajectories search. In: Proceedings of the 7th Workshop on Geographic Information Retrieval, GIR 2013, Orlando, Florida, USA, pp. 27–34 (November 5, 2013)
4. Korn, F., Muthukrishnan, S.: Influence sets based on reverse nearest neighbor queries. SIGMOD Rec. 29(2), 201–212 (2000)
5. Safar, M., Ibrahimi, D., Taniar, D.: Voronoi-based reverse nearest neighbor query processing on spatial networks. Multimedia Systems 15(5), 295–308 (2009)
6. Shang, S., Yuan, B., Deng, K., Xie, K., Zhou, X.: Finding the most accessible locations: reverse path nearest neighbor query in road networks. In: GIS 2011, pp. 181–190 (2011)
7. Stanoi, I., Agrawal, D., Abbadi, A.E.: Reverse nearest neighbor queries for dynamic databases. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 44–53 (2000)
8. Tran, Q.T., Taniar, D., Safar, M.: Reverse *k* nearest neighbor and reverse farthest neighbor search on spatial networks. In: Hameurlain, A., Küng, J., Wagner, R. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems I. LNCS, vol. 5740, pp. 353–372. Springer, Heidelberg (2009)
9. Wang, Y., Xu, C., Gu, Y., Chen, M., Yu, G.: Spatial query processing in road networks for wireless data broadcast. Wirel. Netw. 19(4), 477–494 (2013)
10. Yang, C., Lin, K.-I.: An index structure for efficient reverse nearest neighbor queries. In: ICDE, pp. 485–492 (2001)
11. Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse nearest neighbors in large graphs. IEEE Transactions on Knowledge and Data Engineering 18(4), 540–553 (2006)