# Reverse Nearest Neighbors in Large Graphs

Man Lung Yiu
Department of Computer Science and Information Systems
University of Hong Kong
Pokfulam Road, Hong Kong
*mlyiu2@csis.hku.hk*

Dimitris Papadias
Department of Computer Science
Hong Kong University of Science and Technology
Clearwater Bay, Hong Kong
*dimitris@cs.ust.hk*

Nikos Mamoulis
Department of Computer Science and Information Systems
University of Hong Kong
Pokfulam Road, Hong Kong
*nikos@csis.hku.hk*

Yufei Tao
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Hong Kong
*taoyf@cityu.edu.hk*

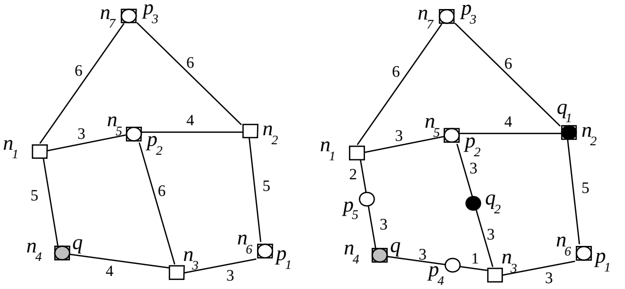
## Abstract

A reverse nearest neighbor query returns the data objects that have a query point as their nearest neighbor. Although such queries have been studied quite extensively in Euclidean spaces, there is no previous work in the context of large graphs. In this paper, we propose algorithms and optimization techniques for RNN queries by utilizing some characteristics of networks.

## Introduction-Motivation

Given a multi-dimensional dataset $P$ and a point $q$, a (*monochromatic*) *reverse nearest neighbor* (RNN) query retrieves all the points $p \in P$ that have $q$ as their nearest neighbor. Given two datasets $P$ and $Q$ and a point $q$, a *bichromatic* (*b*RNN) query retrieves all the points $p \in P$ that are closer to $q$ than to any point of $Q$. The problem has received considerable attention the last few years [KM00, SRAA01, TPL04] due to its relevance in several applications involving decision support, resource allocation, profile-based marketing, etc. However, all the existing work focuses exclusively on Euclidean spaces, whereas in several domains the data are represented as large disk-based graphs.

As an example, consider a P2P network, where each point (i.e., peer) $p \in P$ lies on a node $n \in V$, but some nodes may not contain relevant peers to a specific query. We call such graphs *restricted*. For instance, in Figure 1a, assume that a new user $q$ interested in music enters the system. A (monochromatic) RNN query retrieves among the existing users also interested in music ($p_1$ to $p_3$), the ones for which $q$ will become their new NN. On the other hand, nodes such as $n_1$ and $n_2$ are irrelevant to the user (e.g., they may represent peers with other types of content) and they are considered empty. Given the edge costs of Figure 1a, RNN($q$) = $\{p_1,p_2\}$, i.e., $q$ is most beneficial to $p_1$ and $p_2$ since it is their closest NN in terms of network cost and shares the same interests.



(a) RNN in restricted graph (b) *b*RNN in unrestricted graph

**Figure 1** Examples of RNN queries in graphs

In *unrestricted* networks, the query and the data points can reside anywhere on the edges of the graph. Figure 1b shows an example of a bichromatic query in a road network, where points $p_1$ to $p_5$ stand for residential blocks and $q_1$, $q_2$ indicate restaurants. Nodes $n_1$ and $n_3$ are empty road junctions (i.e., they do not contain residential blocks or restaurants). Given several choices for the location of a new restaurant, the query *b*RNN($q$) may be used to evaluate the benefit of $q$ in terms of the customers that it may attract from rival restaurants based on proximity. Specifically, the set *b*RNN($q$) = $\{p_4,p_5\}$ represents the blocks that are closer to $q$ than to any other competitor. Similarly, *b*RNN($q_1$) = $\{p_1,p_3\}$ and *b*RNN($q_2$) =$\{p_2\}$.

## Algorithms-Optimizations

We propose algorithms for monochromatic and bichromatic queries involving an arbitrary number $k$ of RNN*s* in restricted or unrestricted graphs. Due to space constraints, in the following description we assume restricted networks and that $k$=1. Our algorithms use the following lemma for pruning the search space.

**Lemma 1** Let $q$ be a query point, $n$ a graph node and $p$ a data point satisfying $d(q,n) > d(p,n)$. For any point $p' \neq p$ whose shortest path to $q$ passes through $n$, it holds that $d(q,p') > d(p,p')$, i.e., $p' \notin$ RNN($q$).

**Proof.** $d(q,p') = d(q,n)+d(n,p') > d(p,n)+d(n,p') \geq d(p,p')$. □

For instance, in Figure 1a, $d(q,n_3)=4 > d(p_1,n_3)=3$. Thus, any data point (other than $p_1$) whose shortest path to $q$ passes from $n_3$ cannot be a RNN of $q$ because it is closer to $p_1$ (than $q$). On the other hand, $p_1 \in$ RNN($q$) if there is not other data point within distance $d(p_1,q)$ from $p_1$ (which is true in this example).

The *eager* algorithm initializes a heap $H$ and inserts the source node. When a node $n$ is de-heaped, *eager* applies Lemma 1 in order to determine whether the expansion should proceed. In particular, it first retrieves the NN of $n$ by expanding the network around $n$ (we use a technique similar to Dijkstra's algorithm). If no data point is discovered within distance $d(n,q)$ from $n$, the algorithm en-heaps the adjacent nodes of $n$. If there is a point $p$, such that $d(n,q)>d(n,p)$, the expansion does not proceed further because (according to Lemma 1) $n$ cannot lead to a RNN of $q$. In this case, however, we need to verify (by another network expansion around $p$) if $p \in$ RNN($q$), in which case $p$ is added to the result. Furthermore, $p$ is marked as *verified* in order not to be expanded, if it is found again in the future through another node.

As an example consider the RNN query of Figure 1a initiated at node $n_4$, which is inserted into $H$=<$n_4$,0>. Then, $n_4$ is de-heaped and its adjacent nodes are added to $H$=<$n_3$,4>,<$n_1$,5>. The subsequent removal (from $H$) of $n_3$ triggers an expansion around $n_3$ for finding potential data points closer than the query. In this case, $d(n_3,p_1)=3 < d(n_3,q)$, meaning (by Lemma 1) that we do not search farther, i.e., the adjacent nodes of $n_3$ are not inserted to $H$. Since $q$=NN($p_1$), $p_1$ is an actual result. Next, $n_1$ is de-heaped, its NN $p_2$ is discovered and the search for RNNs terminates because $d(n_1,q)=5 > d(n_1,p_2)=3$. The final step simply verifies $p_2$ as an actual result because $q$=NN($p_2$).

Although *eager* minimizes the number of nodes inserted into the heap, it may perform numerous local network expansions for (i) retrieving the nearest point $p$ of a de-heaped node $n$ and (ii) for verifying whether $p \in$ RNN($q$). The *lazy* algorithm delays pruning until a point is visited. In particular, assuming single RNN retrieval, when the de-heaped node $n$ does not contain a point, *lazy* simply inserts its adjacent nodes into $H$. If $n$ contains a point $p$, the expansion stops since every subsequent node is closer to $p$ than to $q$. Figure 2 illustrates an example where the first data point ($p_1$) is discovered when node $n_4$ is de-heaped. Unlike *eager*, *lazy* does not have to retrieve the NN of $n_4$. However, it still has to verify whether $p_1 \in$ RNN($q$) by an expansion that visits nodes $n_5$, $n_2$ and $n_3$ before determining that $q$=NN($p_1$).

*Lazy* takes advantage of verification queries to prune the search space. Specifically, let $n$ be a node visited by the verification phase of a data point $p$. If $n$ has not been visited by the expansion around the query, it means that $d(n,p)<d(n,q)$ since nodes are visited in ascending order of their distances and the verification query has maximum range $d(p,q)$ (i.e., $d(n,p)<d(p,q)\leq d(n,q)$). Therefore, (by Lemma 1) $n$ cannot lead to any RNNs of $q$. For instance, in Figure 2 the verification of $p_1$ will encounter $n_5$; thus, when $n_5$ is de-heaped later it will be immediately discarded. On the other hand, if $n$ has already been visited by the expansion around $q$, we compare the distances $d(n,p)$ and $d(n,q)$. If $d(n,p) < d(n,q)$, all nodes that were inserted into $H$ by the expansion of $n$ can also be eliminated. In Figure 2, since $d(n_2,p_1) < d(n_2,q)$, node $n_7$ (inserted during the processing of $n_2$) cannot lead to a RNN. Similarly, $n_6$ (inserted during the processing of $n_3$) does not need to be expanded because $d(n_3,p_1) < d(n_3,q)$ and *lazy* terminates with RNN($q$)={$p_1$}.

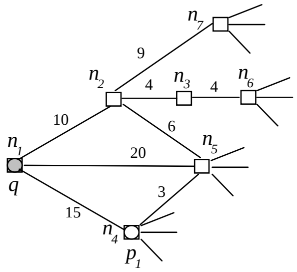


**Figure 2** Example of single RNN by *lazy*

In addition, we propose two optimized versions of the basic algorithms. In particular, *eager-M* materializes the $K$NNs of each node $n$, where $K$ is the maximum number of RNNs to be requested by any query (typically, $K$<<|$P$| where |$P$| is the dataset cardinality). This choice supports efficient updates in the presence of object insertions and deletions. The *lazy-EP* algorithm (*EP* stands for extended pruning) expands the network in parallel using an additional heap that applies the pruning effect of the discovered points. All versions can be extended for arbitrary values of $k$ and retrieval of bichromatic queries.

Extensive experiments with various datasets show that the problem characteristics have a significant effect on the behavior of the algorithms. *Lazy* has lower CPU overhead than *eager* in most settings (therefore, it may be preferable in the presence of large buffers), but it is very expensive for graphs (i.e., computer networks) that incur exponential expansion. *Lazy-EP* usually provides improvements over the basic algorithm. *Eager* has a more balanced behavior than both *lazy* and *lazy-EP*, but the best choice for all settings is *eager-M*, which, however, requires the materialization of the NN points of all nodes.

## References

[KM00] Korn, F., Muthukrishnan, S. Influence Sets Based on Reverse Nearest Neighbor Queries. *SIGMOD*, 2000.

[SRAA01] Stanoi, I., Riedewald, M., Agrawal, D., Abbadi, A. Discovery of Influence Sets in Frequently Updated Databases. *VLDB*, 2001.

[TPL04] Tao, Y., Papadias, D. Lian, X. Reverse *k*NN Search in Arbitrary Dimensionality. *VLDB*, 2004.