

Proximity Queries in Large Traffic Networks

Hans-Peter Kriegel, Peer Kröger, Peter Kunath, Matthias Renz, Tim Schmidt

Ludwig-Maximilians-Universität München, Oettingenstr. 67, 80538 Munich, Germany

WWW: <http://www.dbs.ifi.lmu.de>

Email: {kriegel,kroegerp,kunath,renz,schmidtti}@dbs.ifi.lmu.de

ABSTRACT

In this paper, we present an original network graph embedding to speed-up distance-range and k -nearest neighbor queries in (weighted) graphs. Our approach implements the paradigm of filter-refinement query processing and can be used for proximity queries on both static as well as dynamic objects. In particular, we present how our embedding can be used to compute a lower and upper bounding filter distance which approximates the true shortest path distance significantly better than traditional filters, e.g. the Euclidean distance. These distance approximations can be used within a filter step to prune true drops and true hits as well as in the refinement step in order to guide an informed A* search. Our experimental evaluation on several real-world data sets demonstrates a significant performance boosting of our proposed concepts over existing work.

Categories and Subject Descriptors

H.3.3 [INFORMATION STORAGE AND RETRIEVAL]: Information Search and Retrieval

General Terms

Performance

Keywords

Proximity queries, traffic networks, network embedding

1. INTRODUCTION

Efficient support of proximity queries in large traffic networks are required in many applications of GIS such as location-based services, and traffic network monitoring. Traffic networks are usually modeled by graphs. Nodes of the graph represent crossings such as road intersections or junctions, whereas edges represent connections such as roads or railways between nodes. The data objects representing

points of interest such as cars and service stations are distributed over this road network, i.e. are located at nodes or on edges or may move along the graph.

The most prominent proximity queries in road networks are distance range queries (DRQ) and k -nearest neighbor queries (k NNQ). A DRQ retrieves for a given query object q all static and dynamic objects within the network that have a distance of at most ε to q in the moment of query launching. A k NNQ returns the $k \geq 1$ objects having the smallest distance to a query object q in the moment of query launching. The distance between objects in the network is measured by means of the shortest path distance which can be computed by the well-known Dijkstra algorithm.

Since the Dijkstra algorithm suffers from high computational cost, a lot of research work has been done recently to speed up the network distance computation. However, in today's applications of GIS usually a high number of online queries on networks of hundreds of thousands or even millions of nodes have to be answered in real-time. Obviously, an efficient solution is utterly necessary for such scenarios. Thus, since the distance computation is rather costly, a filter/refinement approach is envisioned, applying a cheaper filter step in order to efficiently partition the data objects into a set of true hits and/or true drops, and a set of candidates, that need to be further analyzed. In order to decide about true hits, we need a progressive filter step that implements the upper bounding property. On the other hand, a conservative filter implementing the lower bounding property is needed to decide about true drops. The remaining set of candidates that cannot be discarded from or included in the result set by means of the filter step, need to be refined, i.e. the true network distance needs to be computed.

In this paper, we propose a novel method for efficient similarity search in large graph networks implementing a filter/refinement architecture. The proposed method is based on a network graph embedding that transforms each graph node into a k -dimensional vector space. We outline how this embedding can be computed and managed efficiently for static and dynamic objects. Based on the embedding, we derive accurate lower- and upper bounds for the network distance which can be computed efficiently. These distance approximations can be used in a filter step to efficiently prune true hits and true drops without exact distance computation. Furthermore, our distance approximations can be used to guide an informed A* search to accelerate shortest path computation in the refinement step.

The paper is organized as follows. Section 2 introduces preliminary definitions and discusses related work. Section

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACM GIS '07, November 7-9, Seattle, WA

Copyright 2007 ACM ISBN 978-1-59593-914-2/07/11...\$5.00.

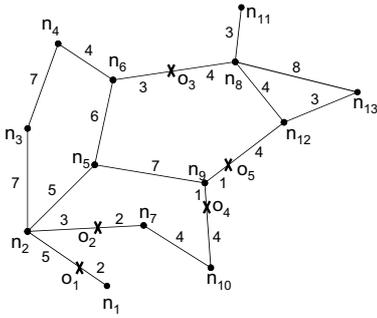


Figure 1: Network graph example.

3 introduces the embedding technique which we call *reference node embedding*. In Section 4 we sketch our multi-step query processing architecture for proximity queries. Section 5 presents a comparative evaluation of the proposed methods and Section 6 concludes the paper.

2. PRELIMINARIES AND RELATED WORK

2.1 Preliminaries

Let \mathcal{D} be a database of objects that are located in a traffic network, e.g. cars or pedestrians in a network of streets. The traffic network is represented by an undirected weighted graph $\mathcal{G} = (N, E, W)$ called *network graph*, where N denotes the set of nodes, $E \subseteq N \times N$ denotes the set of edges and the function $W : E \rightarrow \mathbb{R}^+$ associates a *weight* $w(n_i, n_j)$ to each edge $(n_i, n_j) \in E$. The *network distance* between two nodes $n_i, n_j \in N$, denoted by $d_{net}(n_i, n_j)$, equals $w(n_i, n_j)$ if $(n_i, n_j) \in E$, else it equals the length of the shortest path from n_i to n_j . The length of a path is defined as the sum of the weights of all participating edges.

We assume that each data object $o \in \mathcal{D}$ is located somewhere in the traffic network, i.e. either on a node or on an edge. If an object o is located on an edge $(n_i, n_j) \in E$, $d_i(o)$ and $d_j(o)$ denote the distance of o to the adjacent nodes n_i and n_j , respectively. The network distance between two objects $o_i, o_j \in \mathcal{D}$, $d_{net}(o_i, o_j)$, is the length of the shortest path between o_i and o_j . Thereby, we assume that o_i and o_j are additional “virtual” nodes of the graph. Thus, if o_i is located on edge (n_{i_1}, n_{i_2}) we introduce additional “virtual” edges (o_i, n_{i_1}) and (o_i, n_{i_2}) with weights $w(o_i, n_{i_1}) = d_{i_1}(o_i)$ and $w(o_i, n_{i_2}) = d_{i_2}(o_i)$, respectively. If o_i is located on a node n , we do not need to introduce additional edges or nodes but can work with n instead of o_i . Note, that by introducing the additional “virtual” nodes for objects, the network distance is still a function $N \times N \rightarrow \mathbb{R}$. Whenever we use d_{net} as a function on $\mathcal{D} \times \mathcal{D}$ in the following, we assume the introduction of virtual nodes for the according objects if necessary.

An example network graph is shown in Figure 1, where the graph nodes are depicted as points and the crosses depict the objects. If there is no object on an edge, the corresponding weight is depicted. Else, only the distances of the objects to the adjacent nodes are shown. In this case, the edge weights are obtained by summing up these distances. For example, the weight of edge (n_3, n_2) is $w(n_3, n_2) = 7$ units, whereas the weight of the edge (n_2, n_1) is $d_2(o_1) + d_1(o_1) = 5 + 2 = 7$ units. In our example, the shortest path between the nodes n_{10} and n_6 is $\langle n_{10}, n_D, n_9, n_5, n_6 \rangle$ and has a length of 18

units, i.e. $d_{net}(n_{10}, n_6) = 18$. The shortest path between the objects o_1 and o_3 is $\langle o_1, n_2, n_5, n_6, o_3 \rangle$ and has a length of 19 units, i.e. $d_{net}(o_1, o_3) = 19$.

Based on the network distance, proximity queries are given as follows. Given a query object q located on the graph \mathcal{G} and a distance threshold $\varepsilon \in \mathbb{R}^+$, a *DRQ* returns the set $DRQ(q, \varepsilon) = \{o \in \mathcal{D} \mid d_{net}(q, o) \leq \varepsilon\}$. Given a query object q located on the graph \mathcal{G} and a number $k \in \mathbb{N}^+$, a *kNNQ* returns the set $NNQ(q, k)$ containing k objects such that $\forall o \in NNQ(q, k), \hat{o} \in \mathcal{D} \setminus NNQ(q, k) : d_{net}(q, o) \leq d_{net}(q, \hat{o})$.

2.2 Related Work

Proximity queries in traffic networks are based on network distances defined by the shortest path between two objects. For computing the shortest path, commonly the well-known Dijkstra algorithm [5] is used. It expands the path from the starting node towards the target node using a priority queue of visited nodes sorted by ascending distance from the starting node. Several variants of this algorithm [4] differ on how they manage the priority queue. The A* algorithm [14] applies heuristics to prune the search space and direct the graph expansion. Materialization techniques accelerate shortest path processing by using pre-computed results stored in materialized views [1, 9, 11] but suffer from increasing storage costs. The performance of secondary-memory adaptations of shortest path algorithms has been analyzed in [10, 21]. In [15] the authors divide the graph into regions and gather information whether an edge is on a shortest path leading to a specific region. All these approaches provide only a speed-up for the exact distance computation but cannot be used as a filter step.

Since the network graph corresponds to a traffic network stretching a 2D plane, we can assign to each node and to each object a 2D coordinate w.r.t. an origin. The 2D coordinates of a node n are denoted by $(n.x, n.y)$. Based on these 2D coordinates, the Euclidean distance $d_{eucl}(n_i, n_j) = \sqrt{|n_i.x - n_j.x|^2 + |n_i.y - n_j.y|^2}$ between nodes and/or objects can be computed, corresponding to the direct “airline distance” between these (possibly virtual) nodes. One of the first papers which deals with the efficient processing of spatial queries is [17] using the Euclidean distance as a lower bounding filter in order to guide an incremental network expansion for refinement. This approach works well for high-proximity queries (i.e. small query range or small nearest-neighbor coefficient k) and dense object distributions. However, when the objects in the traffic network are sparsely distributed, then this approach requires to retrieve a large portion of the network for distance computation and consequently yields a poor performance. Furthermore, as this approach uses the Euclidean distance to guide the network expansion it does not provide an upper bounding filter. Thus, it can only be used to decide about true drops, but not about true hits. This results in a larger amount of candidates that need to be refined. In addition, the lower bound approximation using Euclidean distance is often very coarse.

In [20] one of the graph embedding technique from [16] is applied in order to estimate the network distance between two nodes in a very efficient way. On the basis of the traditional approach they present an extended dynamic embedding which takes into consideration that the objects on the network move, i.e. change continuously their locations. In addition, they show how the graph embedding can be used to compute the shortest path between two objects. How-

ever, they proposed only an algorithm which computes an approximated shortest path, whereas, the accuracy of the query result depends on the density and distribution of the objects in space. Furthermore, the embedded space of the presented technique involves 40 to 256 dimensions, and thus, high-dimensional index structures are required. In our approach, we also use an embedding technique that transforms the planar network graph in a higher dimensional space. Instead of a reference-set-based embedding as proposed in [20], we use a reference-node-based embedding that achieves adequate distance approximations with a low-dimensional embedding space, even for very large traffic networks. The distance approximations obtained from the embedded space can then be used in a filter step to identify the result candidates of the DRQ or k NNQ. The main drawback of the approach presented in [20] is that it does not offer any solution for the computation of the exact distances of the candidates in the subsequent refinement step.

Recently, Hu et al. proposed another solution for the pre-computation and indexing of network distances for objects in road networks [8]. The basic idea of their approach is to use *distance signatures* for each data object in the network graph which are computed offline and stored efficiently. The distance signature of an object o contains a vector of distance approximations between o and all other data objects in the network graph. These distance approximations are then used to efficiently determine the candidates of a proximity query in a filter step. Subsequently, the exact distances of the candidates have to be computed online in the refinement step. This computation is supported by a linked list associated with each distance signature that optimally guides the search of the shortest paths from the query object q to the candidates. The main drawback of this proposal is that the storage and query cost highly depend on the number of data objects, as each data object leads to an entry in the distance signature. For a reasonable number of data objects, the storage cost of the signatures might explode which would lead to a poor query performance. Furthermore, this approach does not support proximity queries on moving objects, i.e. objects frequently changing their actual positions.

An alternative approach to the partial or complete materialization of the network distances is the *solution indexing* method that pre-computes and stores the solutions of the proximity queries. One instance of this technique is the Voronoi cell based k NN approach (VN^3) proposed in [13]. A Voronoi diagram on the network space is computed and each Voronoi cell that represents the region of the nearest neighbor in the network is represented by a two-dimensional polygon. A spatial access method, e.g. the R-tree, is used to index these Voronoi-cell polygons, such that one-nearest neighbor queries are reduced to point queries. Furthermore, it is shown that the network Voronoi diagram can also support k NN queries ($k > 1$) by taking the local neighborhood of the cells into account. However, the performance of the VN^3 approach mainly depends on the density and distribution of the objects in the network. Dense network graphs on which the data objects are sparsely distributed lead to large Voronoi cells with a lot of adjacent neighbor cells. In this case, the computation of the k NN would have a poor performance. Hence, the VN^3 approach is only suitable for reasonably dense datasets and/or sparse network graphs.

In this paper, we do not focus on another class of proximity queries in road networks called continuous proximity

queries (as studied e.g. in [12, 3]). Those queries retrieve the objects satisfying a query condition at any point on the path of a moving query object, e.g. continuous nearest neighbor queries generate a set of path regions and their corresponding k NNs, such that each point on a path region (interval) has the same k NN.

Let us note that the large number of techniques for indexing and querying static or dynamic objects in Euclidean spaces (e.g. [6, 19, 2] and [26, 22, 23, 24, 25]) do not qualify for proximity queries in traffic networks or generally in spaces constituting any motion constraints.

3. NETWORK GRAPH EMBEDDING

3.1 Reference Node Embedding for Static and Dynamic Objects

The basic idea of our approach is to transform the nodes of any network graph and the objects located on that graph into a k -dimensional vector space adapting the Lipschitz embedding technique with singleton reference sets called *reference nodes*.

Let $\mathcal{G} = (N, E, W)$ be a network graph and $N' = \langle n_{r_1}, \dots, n_{r_k} \rangle$ a subsequence of nodes, $N' \subseteq N$ containing $k \geq 1$ nodes called *reference nodes*. The metric space (N, d_{net}) is called *native space* throughout the paper. The embedding, or transformation, of the native space (N, d_{net}) into a k -dimensional *vector space* (\mathbb{R}^k, D) is a mapping $F^{N'} : N \cup \mathcal{D} \rightarrow \mathbb{R}^k$, where $|N'| = k$ is the dimensionality of the vector space and D is the L_∞ -norm in \mathbb{R}^k , i.e. $D(x, y) = \max_{i=1..k} |x_i - y_i|$, where $x, y \in \mathbb{R}^k$ are two points of the vector space (\mathbb{R}^k, D) . A *reference node embedding* of \mathcal{G} based on a set of reference nodes $N' \subset N$ defines the function $F^{N'}$ as follows.

1. For each node $n \in N$,

$$F^{N'}(n) = (F_1^{N'}(n), \dots, F_k^{N'}(n))^T,$$

where $F_i^{N'}(n) = d_{net}(n, n_{r_i})$ for $1 \leq i \leq k$.

2. For each object $o \in \mathcal{D}$ located on a node n , $F^{N'}(o) = F^{N'}(n)$.

3. For each object $o \in \mathcal{D}$ located on an edge $(n_1, n_2) \in E$,

$$F^{N'}(o) = (\hat{F}_1^{N'}(o), \dots, \hat{F}_k^{N'}(o))^T,$$

where $\hat{F}_i^{N'}(o) = \min\{d_1(o) + F_i^{N'}(n_1), d_2(o) + F_i^{N'}(n_2)\}$.

For the embedding of a network graph we have to compute for each node and each object the shortest paths to all reference nodes. As long as the graph structure and the positions of the objects do not change, this operation has to be performed only once in a preprocessing step. If we want to handle dynamic objects, a re-embedding of the objects would be necessary. In fact, if we assume that the graph structure remains fixed (which is obviously a realistic assumption) and we store the embedding of the graph nodes (that do not change), the re-embedding of the objects after each re-positioning can be done rather efficiently. We can use the pre-computed embedding of the graph nodes in order to incrementally update the embedding of the dynamic objects. The computation of the components of the embedding vector $F^{N'}(o)$ of an object $o \in \mathcal{D}$ located on an edge $(n_1, n_2) \in E$ is given by the functions $\hat{F}_i^{N'}(o)$. These values

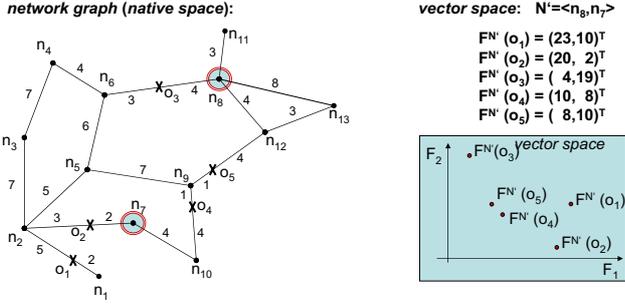


Figure 2: Network graph embedding.

can be very efficiently computed assuming that $F^{N'}(n_1)$ and $F^{N'}(n_2)$ is given. For this reason, the embedding function $F^{N'}$ is very suitable for both static and dynamic objects.

In Figure 2 we demonstrate the embedding of the network graph depicted in Figure 1 using the nodes n_8 and n_7 as reference nodes, i.e. $N' = \langle n_8, n_7 \rangle$. On the left hand side, the original network graph (native space) including the object (depicted with crosses) and the reference nodes (emphasized by circles) are visualized. On the right hand side, the transformation of the objects in the vector space using the embedding function $F^{N'}$ is shown.

3.2 Distance Estimation

For the sake of simplicity, we use the term “nodes” for both graph nodes and objects located somewhere in the graph, i.e. for the elements of the set $N \cup \mathcal{D}$ in the following. We simply assume that additional “virtual” nodes are introduced for each object that is located on an edge as described above.

The reference node embedding can be used to compute upper and lower bounds for the network distance. In fact, the embedding function $F^{N'}$ already provides a conservative approximation of the network distance, i.e. the distance D in the embedded space lower bounds the distance d_{net} in the native space, formally:

LEMMA 1 (LOWER BOUNDING PROPERTY).

Let $\mathcal{G} = (N, E, W)$ be a network graph, $N' \subset N$ be a set of reference nodes, and $F^{N'}$ be the reference node embedding w.r.t. N' . For any two nodes $n_i, n_j \in N \cup \mathcal{D}$, the following statement holds:

$$D(F^{N'}(n_i), F^{N'}(n_j)) \leq d_{net}(n_i, n_j).$$

PROOF. Let $N' = \langle n_{r_1}, \dots, n_{r_k} \rangle$. Since the network distance is transitive, the following statements hold:

$$D(F^{N'}(n_i), F^{N'}(n_j)) = \max_{l=1, \dots, k} |F_l^{N'}(n_i) - F_l^{N'}(n_j)| =$$

$$\max_{l=1, \dots, k} |d_{net}(n_i, n_{r_l}) - d_{net}(n_j, n_{r_l})| \leq d_{net}(n_i, n_j)$$

□

The embedded space can also be used to define a progressive approximation of the network distance. In particular, the distance function D^* , which is defined as

$$D^*(x, y) = \min_{i=1, \dots, k} (x_i + y_i),$$

is an upper bound of d_{net} , formally

LEMMA 2 (UPPER BOUNDING PROPERTY).

Let $\mathcal{G} = (N, E, W)$ be a network graph, $N' \subset N$ be a set of reference nodes, and $F^{N'}$ be the reference node embedding w.r.t. N' . For any two nodes $n_i, n_j \in N \cup \mathcal{D}$, the following statement holds:

$$D^*(F^{N'}(n_i), F^{N'}(n_j)) \geq d_{net}(n_i, n_j).$$

PROOF. Let $N' = \langle n_{r_1}, \dots, n_{r_k} \rangle$. Due to the transitivity of the network distance, the following statements hold:

$$D^*(F^{N'}(n_i), F^{N'}(n_j)) = \min_{l=1, \dots, k} (F_l^{N'}(n_i) + F_l^{N'}(n_j)) =$$

$$\min_{l=1, \dots, k} (d_{net}(n_i, n_{r_l}) + d_{net}(n_j, n_{r_l})) \geq d_{net}(n_i, n_j)$$

□

In summary, the reference node embedding allows the definition of an upper bound D^* and a lower bound D of the true network distance that can be used in a filter/refinement query processing architecture.

3.3 Choosing the Reference Node Set

Obviously, the quality of our approximation distances D and D^* , i.e. the approximation error w.r.t. the network distance d_{net} , depends on the number and location of the used reference nodes. In this subsection we discuss a suitable location of the reference nodes.

Let $\mathcal{P}_{best}(n_s, n_d)$ be the shortest path between two nodes $n_s, n_d \in N \cup \mathcal{D}$. Obviously, for each node $n_i \in N \cup \mathcal{D}$ in $\mathcal{P}_{best}(n_s, n_d)$, the shortest path from n_i to n_d , $\mathcal{P}_{best}(n_i, n_d)$, is a subsequence of $\mathcal{P}_{best}(n_s, n_d)$. By means of this property we can identify those object pairs for which the distance estimation based on a specific reference node set $N' \subseteq N$ is equal to the exact network distance. Let $(n_i, n_j) \in (N \cup \mathcal{D})^2$ be a pair of nodes for which we want to estimate the network distance based on the reference node set N' , then the following two statement holds:

- (1) If a reference node $n_r \in N'$ exists for which $n_j \in \mathcal{P}_{best}(n_i, n_r)$ or $n_i \in \mathcal{P}_{best}(n_j, n_r)$ then $D(F^{N'}(n_i), F^{N'}(n_j)) = d_{net}(n_i, n_j)$ holds (e.g. object pair (o_1, o_2) in Figure 2).
- (2) If a reference node $n_r \in N'$ exists for which $n_r \in \mathcal{P}_{best}(n_i, n_j)$ then $D^*(F^{N'}(n_i), F^{N'}(n_j)) = d_{net}(n_i, n_j)$ holds (e.g. node pair (o_1, o_4) in Figure 2).

Hence, the first statement argues for selecting nodes as reference nodes that are located at the outer margin of the network graph. The second statement would argue to choose nodes which are more centrally located in the network as reference nodes. In general, we can reduce the distance approximation error by locating the reference nodes as close as possible to the data objects, such that each object is nearby to at least one reference node. In the static case, this can be achieved by clustering the object nodes using a k-medoid algorithm and take the cluster medoids as reference nodes. Assuming dynamic objects, we have to adapt the reference nodes to the average object distribution, e.g. the city center is a potential hotspot where permanently a lot of objects are located. If we do not have the chance to identify such hotspots, we suggest to distribute the set of reference nodes equally over the network graph.

```

DRQ( $q, \varepsilon, \mathcal{G}$ )
  candidateSet :=  $\emptyset$ ;
  resultSet :=  $\emptyset$ ;
  /**** FILTER STEP *****/
  for each  $o \in \mathcal{D}$  do
    if  $D(F^{N'}(q), F^{N'}(o)) \leq \varepsilon$  then
      if  $D^*(F^{N'}(q), F^{N'}(o)) \leq \varepsilon$  then
        add  $o$  to resultSet;
      else add  $o$  to candidateSet;
  /**** REFINEMENT STEP *****/
  for each  $o \in$  candidateSet do
    if  $d_{net}(q, o) \leq \varepsilon$  then
      add  $o$  to resultSet;
  return resultSet;

```

Figure 3: DRQ algorithm.

4. MULTI-STEP QUERY PROCESSING

In Section 3 we have shown that our reference node embedding approach provides an upper and a lower bounding approximation for the exact network distance. Thus, our approach can be successfully applied to a multi-step query processing in order to efficiently support DRQ and k NNQ in large network graphs. In the following, we present the multi-step DRQ and k NNQ using our embedding function $F^{N'}$ implementing a reference node embedding. As mentioned above, for static objects, the graph embedding has to be performed only once in a preprocessing step before any query is launched. In case of dynamic objects, the re-embedding can be computed rather efficiently on the fly as sketched in Section 3. For the refinement, we apply an A* search using our distance approximations as heuristics to guide the search rather than the traditional Euclidean distance. Since our approximations are more accurate, even the refinement step will be accelerated using our concepts.

DRQ. We first perform the filter step by applying a DRQ over the embedded objects and nodes. All objects for which the conservative distance approximation D is greater than ε can be discarded as true drops without refining them. Furthermore, we can immediately add all objects to the result list if the distance estimation D^* is lower or equal to ε . Only the remaining candidates need to be refined by computing the exact shortest path distance to the query. Obviously, the filter step should be very fast because D and D^* can be computed very efficiently. The pseudocode for a DRQ is given in Figure 3.

k NNQ. We adapt the multi-step nearest-neighbor algorithm proposed in [18] which is shown to be optimal w.r.t. the number of candidates that are refined. Since this algorithm can only use a lower bound for the filter step but is not designed to consider an upper bound, we integrate our upper bounding distance approximation by a simple trick. The resulting algorithm is described in the following, the pseudocode is depicted in Figure 4. At first, a ranking query [7] RQ is initialized on the embedded objects, which incrementally reports the objects o in ascending order of the lower bounding filter distance $D(F^{N'}(q), F^{N'}(o))$. Then, we initially fetch the first k candidates from RQ . At this point we slightly adapt the algorithm proposed in [18]. Before refining the first k candidates we report those candidates as true results having an upper bounding distance D^* to the query

```

kNNQ( $q, k, \mathcal{G}$ )
  initialize ranking :=  $RQ(q, \mathcal{D})$ ;
  resultSet := new list(dist, objectID) sorted by dist;
   $d_{max} := \infty$ ;
  /**** FILTER STEP (different to [18]) *****/
  candidateSet :=  $\emptyset$ ;
  for first  $k$  objects  $o \in$  ranking do
    candidateSet.insert( $o$ );
     $d_{max} := D(F^{N'}(candidateSet.get(k)), F^{N'}(q))$ ;
  for each  $o \in$  candidateSet do
    if  $D^*(F^{N'}(o), F^{N'}(q)) \leq d_{max}$  then
      add  $(D^*(F^{N'}(o), F^{N'}(q)), o)$  to resultSet;
  /**** ITERATIVE REFINEMENT (see [18]) *****/
  while  $o =$  ranking.getNext() and  $D(o, q) \leq d_{max}$  do
    if  $d_{net}(o, q) \leq d_{max}$  then
      add  $(d_{net}(o, q), o)$  to resultSet;
    if |resultSet|  $\geq k$  then
       $d_{max} =$  resultSet[ $k$ ].dist;
    remove all entries from resultSet with dist  $> d_{max}$ ;
  return resultSet;

```

Figure 4: k NNQ algorithm.

smaller than the lower bounding distance D of the k -th candidate. Afterwards our algorithm proceeds like the original one performing an iterative refinement as long as the lower bound of the next object in the ranking is smaller or equal to the current k -th nearest neighbor distance d_{max} .

5. EVALUATION

For our experiments we used real road networks of San Joaquin County (“TG”, approx. 18,300 nodes), San Francisco (“SA”, approx. 175,000 nodes), and the United States (“NA”, approx. 176,000 nodes). The network objects were simulated through randomized subsets of the graph nodes. As explained in Section 3, object nodes can be easily embedded online. The graph was stored on disk implementing the approach proposed in [17] with a block size of 8 KB and an average storage load of 71% in order to simulate a dynamic environment where the disk blocks are usually not completely filled. Following [17], we organized the graph in three R-trees where the first manages the nodes, the second the edges and the third the polylines, i.e. the detailed representations of all segments in the network. An embedding vector is considered as a further attribute of a node, similar to e.g. the node coordinates, and stored along with it. The two other R-trees holding the edges and the polylines do not need to be modified for our new approach. The reference nodes were chosen by spatially ordering all graph nodes along a Hilbert curve. We then uniformly distributed the reference nodes along this curve. Datasets without an embedding are denoted as REF and reference node embeddings are denoted as 1RNE. All experiments were performed on a workstation featuring a 1.8 GHz CPU, 2GB RAM, a random disk page access time of 6 ms, and a transfer rate of 86MB/s. The cache size was set to 5% of the dataset size. In all experiments, we performed 1,000 random queries and averaged the results.

5.1 Distance Approximation Using Reference Node Embeddings

We first evaluated the quality of the distance approximations gained by our reference node embedding on the TG and

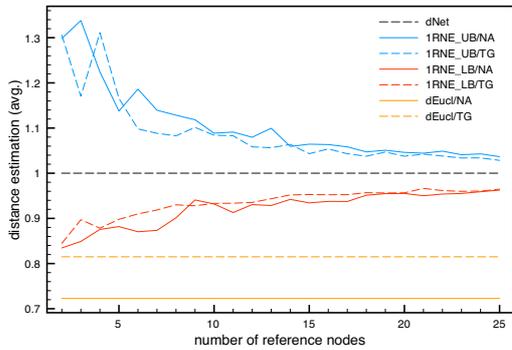


Figure 5: Quality of distance estimations for NA and TG.

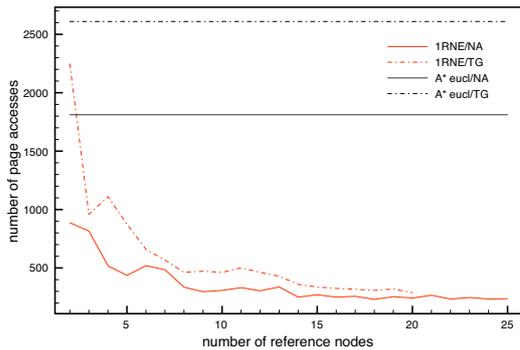


Figure 6: Number of page accesses per distance computation for NA and TG.

the NA datasets. Both datasets differ significantly in their sizes. Figure 5 shows the relative error of the network distance estimation using the Euclidean distance (lower bound) as well as D (lower bound) and D^* (upper bound) w.r.t. the number of reference nodes used for the embedding. Obviously, our lower bounding distance estimation D approximates the true network distance far better than the Euclidean distance. In addition, it turns out that also the upper bound is an accurate approximation of the true distance. In addition, the results suggest – as expected – that the quality of the distance approximations gets better with increasing number of reference nodes. However, for both the TG and the NA datasets, the increase of the approximation quality is only small when using more than 10 reference nodes. This is really notable since TG and NA differ significantly in the number of total graph nodes and suggests that using 10 reference nodes is a good choice even for very large graphs.

The good quality of our approximations is confirmed by a further experiment. We computed several sample shortest paths using the original Dijkstra algorithm, the A* algorithm using the Euclidean distance as distance approximation, and the A* algorithm using our novel distance approximations based on a reference node embedding (with 10 reference nodes each). In all cases, the A* algorithm using our novel distance approximations significantly outperformed the other approaches. Figure 6 illustrates the average number of page accesses per shortest path computa-

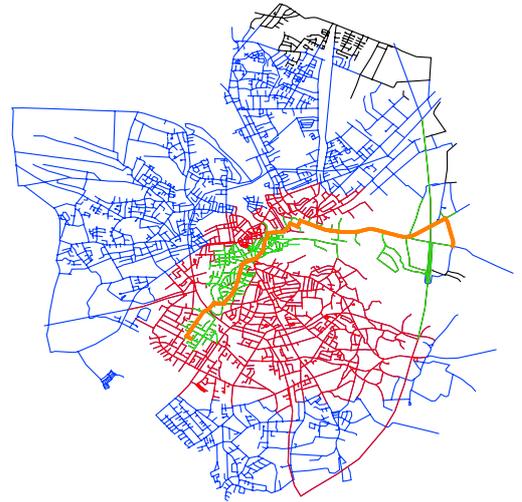


Figure 7: Search space for computing a sample shortest path (marked in orange) with Dijkstra (blue), A* Euclidean (red), and RNE with 10 reference nodes (green).

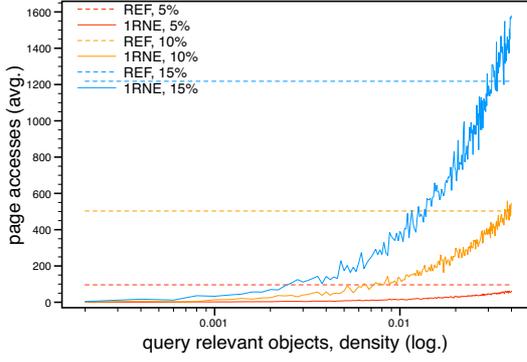
tion on the NA and TG datasets w.r.t. different numbers of reference points. Again it can be seen that choosing 10 reference nodes yields a considerable speed-up over traditional approaches. The reason for the performance boost is illustrated in Figure 7 showing the search space of the three competing shortest path computation approaches for a sample query (best seen in color). As it can be seen, the search space of the A* algorithm using our novel distance approximations is clearly reduced in comparison to the search space of the traditional Dijkstra algorithm and the A* algorithm using the Euclidean distance.

5.2 Filter/Refinement Query Processing

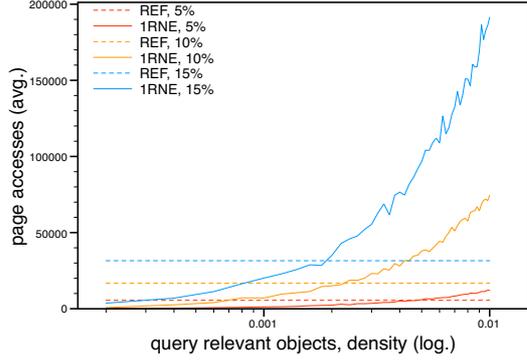
In the following, we evaluate the performance of our novel filter/refinement query processing using a reference node embedding with 10 reference nodes. We examine the performance of the proposed method w.r.t. the relative density of data objects in the network graph, i.e. $|\mathcal{D}|/|N|$.

DRQ. In Figures 8(a) and 8(b) the performance of our filter/refinement query processing for DRQ is shown in terms of the average number of page accesses and number of candidates that need to be refined w.r.t. the object density $\frac{|\mathcal{D}|}{|N|}$. We evaluated DRQs with different query selectivities, in particular, selectivities of 5%, 10%, and 15% of $|\mathcal{D}|$. Let us note that the x-axes in Figures 8(a), and 8(b) are in log scale. Thus, we observe that the number of page accesses as well as the number of candidates scales linear in $\frac{|\mathcal{D}|}{|N|}$. It can also be derived from Figures 8(a) and 8(b), that applying the filter step is more efficient than performing a single-step algorithm using our Shortest Path algorithm (depicted as “REF” in the Figure) even for considerably small densities $|\mathcal{D}|/|N|$.

k NNQ. Similar observations can be made when evaluating our filter/refinement query processing for k NNQ (cf. Figures 9(a) and 9(b)). We varied the number k of nearest neighbors and report results for $k = 5, 10, 20$. As it can be seen,



(a) NA dataset



(b) SF dataset

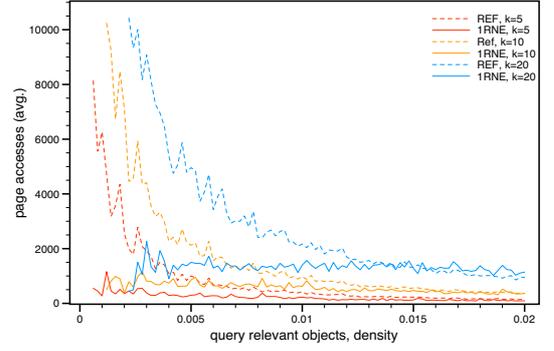
Figure 8: DRQ.

the filter works very well for rather low densities $|\mathcal{D}|/|N|$ producing a small number of candidates. After a “jump” at a certain object density, the number of page accesses as well as the number of candidates again scale linear in $\frac{|\mathcal{D}|}{|N|}$. Please note that the scaling of the y-axis of the SF dataset is logarithmic. The decreasing tendency of the page accesses for both datasets can be explained with the increasing density of the query relevant objects. The higher the density, the less pages have to be accessed in order to locate the k nearest neighbors.

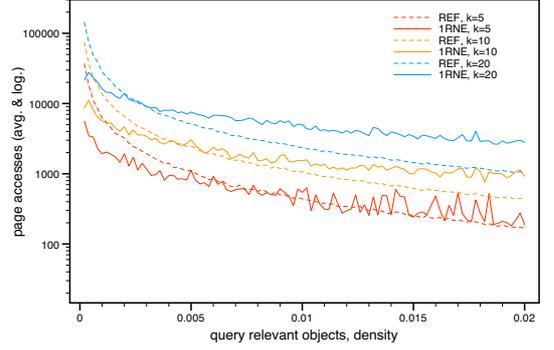
5.3 Comparison with other Approaches

We chose the distance signature (DS) approach [8] as comparison partner because it outperforms other methods such as the network voronoi diagram [13]. The DS method was parameterized as described in [8]. In order to guarantee a fair comparison, we computed two embeddings: one embedding used 30 reference nodes and used 60 reference nodes. For an object density of 0.01 the embedding with 30 reference nodes requires half of the storage space required by the DS approach, whereas the embedding with 60 reference nodes occupies an equal amount of storage space required by the DS approach. Let us note that the object density linearly influences the memory footprint of our embedding technique. In contrast, for the DS approach the relationship between object density and memory consumption is quadratic.

Performance Comparison. The DRQ experiments in



(a) NA dataset



(b) SF dataset

Figure 9: k NNQ.

Figure 10 show that the DS approach is outperformed by all our approaches up to one order of magnitude in terms of the number of required page accesses. In fact, even an embedding with 10 reference nodes clearly outperforms the DS approach while saving more than 3/4 of the storage cost compared to the DS method.

Update Cost Comparison. Due to space limitations, the comparative experiments on the cost of updating the embeddings for moving objects of both approaches are omitted. However, it is worth noting that our approach again significantly outperforms the DS approach. This is due to the fact that the update of the embedding vector of a moving object can be computed rather simple from the embedding of the network graph (see Section 3) and in general needs at most one page access per update. Even if an object moves to a new edge, the embedding of the network nodes remains fixed and only the embedding of the particular object requires re-computation.

5.4 Summary

In summary, our experiments show that our novel embedding yields accurate lower and upper bounds that can be used in a filter/refinement approach to efficiently support distance range and nearest neighbor queries; due to these accurate filter distances, our multi-step query processing significantly outperforms single-step query processing and existing multi-step query processing approaches.

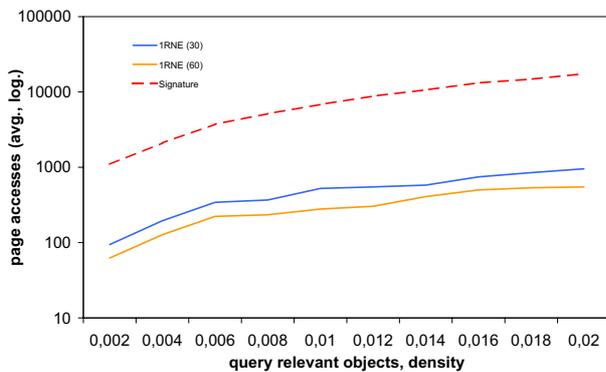


Figure 10: DRQ comparison of RNE and Signature approach (TG dataset).

6. CONCLUSIONS

We proposed a novel graph embedding technique which is suitable for static and dynamic objects using the concept of reference nodes. This embedding provides an efficient and effective upper and lower bound for the network distance and, thus, can be used to implement a filter/refinement architecture for similarity search in large traffic networks. Our experiments show that our novel approach outperforms existing competitors in terms of pruning power in the filter step, thus, requiring far less calls of the Dijkstra (or A*-search) algorithm for exact shortest path computation.

7. REFERENCES

- [1] R. Agrawal, S. Dar, and H. Jagadish. "Direct Transitive Closure Algorithms: Design and Performance Evaluation". *TODS*, 15(3), 1990.
- [2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles". In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'90)*, 1990.
- [3] H.-J. Cho and C.-W. Chung. An efficient and scalable approach to cnn queries in a road network. In *Proc. Int. Conf. on Very Large Databases (VLDB'05)*, 2005.
- [4] T. H. Corman, C. E. Leiserson, and R. L. Rivest. "Introduction to Algorithms". MIT Press, 1990.
- [5] E. W. Dijkstra. "A Note on Two Problems in Connection with Graphs". *Numerische Mathematik*, 1:269-271, 1959.
- [6] A. Guttman. "R-trees a dynamic index structure for spatial searching". In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'84)*, 1984.
- [7] G.R. Hjaltason and H Samet. "Ranking in Spatial Databases". In *Proc. Int. Symp. Large SpatialDatabases (SSD'95)*, 1995.
- [8] H. Hu, D. L. Lee, and V. C. S. Lee. "Distance Indexing on Road Networks". In *Proc. Int. Conf. on Very Large Databases (VLDB'06)*, 2006.
- [9] Y. Ioannidis, R. Ramakrishnan, and L. Winger. "Transitive Closure Algorithms Based on Graph Traversal". *TODS*, 18(3), 1993.
- [10] B. Jiang. "I/O-Efficiency of Shortest Path Algorithms: An Analysis". In *Proc. Int. Conf. on Data Engineering (ICDE'92)*, 1992.
- [11] S. Jung and S. Pramanik. "HiTi Graph Model of Topographical Roadmaps in Navigation Systems". In *Proc. Int. Conf. on Data Engineering (ICDE'96)*, 1996.
- [12] M. Kolahdouzan and C. Shahabi. "Continuous K-Nearest Neighbor Queries in Spatial Network Databases". In *In Proc. of STDBM, 2004*, 2004.
- [13] M. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases". In *Proc. Int. Conf. on Very Large Databases (VLDB'04)*, 2004.
- [14] R. Kung, E. Hanson, Y. Ioannidis, T. Sellis, L. Shapiro, and M. Stonebraker. "Heuristic Search in Data Base Systems". *Expert Database Systems*, 1986.
- [15] E Köhler, R. H. Möhring, and H. Schilling. "Acceleration of Shortest Path and Constrained Shortest Path Computation". In *Proc. Int. WS Efficient and Experimental Algorithms (WEA'05)*, 2005.
- [16] N. Linial, E. London, and Y. Rabinovich. "The geometry of graphs and some of its algorithmic applications". In *Proc. IEEE Symp. Foundations of Computer Science*, 1994.
- [17] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. "Query Processing in Spatial Network Databases". In *Proc. Int. Conf. on Very Large Databases (VLDB'03)*, 2003.
- [18] T. Seidl and H.-P. Kriegel. "Optimal Multi-Step k-Nearest neighbor Search". In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'98)*, 1998.
- [19] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. "The R+-Tree: A Dynamic Index for Multi-Dimensional Objects". In *Proc. Int. Conf. on Very Large Databases (VLDB'87)*, 2000.
- [20] C. Shahabi, M.R. Kolahdouzan, and M. Sharifzadeh. "A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases". *Geoinformatica*, 7(3):255-273, 2003.
- [21] S. Shekhar, A. Kohli, and M. Coyle. "Path Computation Algorithms for Advanced Traveller Information System (ATIS)". In *Proc. Int. Conf. on Data Engineering (ICDE'93)*, 1993.
- [22] Z. Song and N. Roussopoulos. "K-Nearest Neighbor Search for Moving Query Point". In *Proc. Int. Symp. Spatial and Temporal Databases (SSTD'01)*, 2001.
- [23] Y. Tao and D. Papadias. "Time Parameterized Queries in Spatio-Temporal Databases". In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'02)*, 2002.
- [24] Y. Tao, D. Papadias, and Q. Shen. "Continuous NearestNeighbor Search". In *Proc. Int. Conf. on Very Large Databases (VLDB'02)*, 2002.
- [25] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Lee. "Location-based Spatial Queries". In *Proc. ACM Int. Conf. on Management of Data (SIGMOD'03)*, 2003.
- [26] B. Zheng and D. Lee. "Dependent Query Processing". In *Proc. Int. Symp. Large Spatial Databases (SSD'01)*, 2001.