In Proc. 20th Int. Conf. on Scientific and Statistical Database Management (SSDBM), Hong Kong, China, 2008

Hierarchical Graph Embedding for Efficient Query Processing in Very Large Traffic Networks

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

Ludwig-Maximilians-Universität München, Oettingenstr. 67, 80538 Munich, Germany,
{kriegel,kroegerp,renz,schmidtti}@dbs.ifi.lmu.de,
WWW home page: http://www.dbs.ifi.lmu.de

Abstract. We present a novel graph embedding to speed-up distance-range and k-nearest neighbor queries on static and/or dynamic objects located on a (weighted) graph that is applicable also for very large networks. Our method extends an existing embedding called reference node embedding which can be used to compute accurate lower and upper bounding filters for the true shortest path distance. In order to solve the problem of high storage cost for the network embedding, we propose a novel concept called hierarchical embedding that scales well to very large traffic networks. Our experimental evaluation on several real-world data sets demonstrates the benefits of our proposed concepts, i.e. efficient query processing and reduced storage cost, over existing work.

1 Introduction

Similarity queries in large traffic networks are important database operations in applications such as location-based services, traffic network monitoring, traffic information systems, etc. Typically, traffic networks such as road networks are 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, service stations, etc. are distributed over this road network, i.e. are located at nodes or on edges or may move along the graph. The distance between objects in the network is measured by means of the shortest path distance which can be computed by the Dijkstra algorithm.

In today's applications 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, a more efficient solution than computing Dijkstra for all these query nodes is utterly necessary for such scenarios. 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 an upper bounding distance approximation, whereas a lower bounding distance approximation 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.

Here, we propose a novel filter/refinement query processor for very large graph networks based on a hierarchical network graph embedding. Section 2 introduces preliminary definitions and discusses related work. In Section 3, we show how the so-called

reference node embedding can be extended to reduce the storage cost for very large networks. We show how this novel embedding can be computed efficiently for static and dynamic objects located on graph networks and derive efficient lower- and upper bounds for the network distance from the hierarchical embedding. Section 4 sketches our multi-step query processor. Section 5 presents an experimental evaluation of the proposed concepts 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 are adjacent, i.e. $(n_i, n_j) \in E$, 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.

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.

Based on the network distance, proximity queries are given as follows. Given a query object q located on $\mathcal{G}$ and a distance threshold $\varepsilon \in \mathbb{R}^+$, a *distance range query* (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 $\mathcal{G}$ and a number $k \in \mathbb{N}^+$, a *k-nearest neighbor query* (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, e.g. computed by the Dijkstra algorithm [1] and its variants [2]. These algorithms expand 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. The A* algorithm [3] applies heuristics to prune the search space and direct the graph expansion. Materialization techniques [4–6] suffer from increasing

storage cost. In [7] 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.

In [8] the Euclidean distance between graph nodes/objects is used as a lower bounding filter in order to guide an incremental network expansion for refinement. This approach works well only for high-proximity queries (i.e. small query range ε or small nearest-neighbor coefficient k) and dense object distributions, otherwise a large portion of the network for distance computation need to be retrieved. Furthermore, this approach does not provide an upper bounding distance function to filter out true hits, resulting in a larger amount of refinements.

In [9] one of the graph embedding technique from [10] is applied in order to estimate the network distance between two nodes. An extended dynamic embedding for moving objects is presented. In addition, it is shown how the graph embedding can be used to compute an approximate shortest path between two objects. The accuracy of the approximation depends on the density and distribution of the objects in space. A severe drawback of the approach is that the embedded space involves 40 to 256 dimensions. In addition, it does not offer any solution for the computation of the exact distances of the candidates in the refinement step.

In [11] *distance signatures* are computed and managed for each data object o in the network graph containing 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 are computed online in the refinement step. The obvious drawback of this proposal is that the storage and query cost directly depend on the number of objects. Furthermore, this approach does not support an efficient re-embedding necessary to answer proximity queries on moving objects that frequently change their positions.

The work of this paper is based on the network graph embedding originally proposed independently by two research groups [12, 13] and [14]. While the work in [12, 13] only explores a lower bound, the authors in [14] also derive an upper bound for the network distance. In addition, the authors in [12, 13] focus only on speeding up the shortest path computations whereas in [14], the authors propose a multi-step query processing framework for supporting proximity queries in traffic networks. The details of the embedding is reviewed in Section 3.1.

In [15] 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 2D polygon. These Voronoi-cell polygons are indexed to support kNN queries. The performance of this 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 kNN would have a poor performance.

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 [16, 17]).

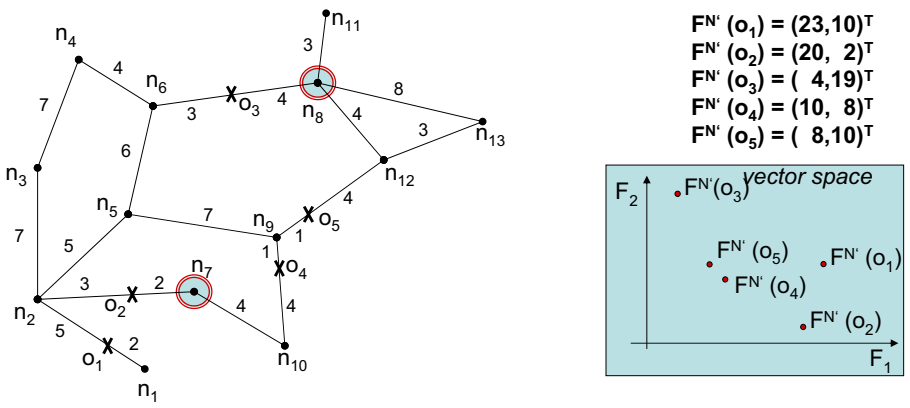


Fig. 1. Network graph embedding.

3 Network Graph Embedding

3.1 Basics

Our approach is based on a special form of a Lipschitz embedding of the traffic network using singleton reference sets which we call *reference nodes* according to [14] (in [12, 13], these reference nodes are called *landmarks*). The embedding transforms the nodes of a given network graph and the objects located on that graph into a k-dimensional vector space. Let $\mathcal{G} = (N, E, W)$ be a network graph and $N' = \langle n_{r_1}, \ldots, n_{r_k} \rangle \subseteq N$ be a subsequence of $k \geq 1$ reference nodes. The embedding, or transformation, of the native space N into a k-dimensional vector space $\mathbb{R}^k$ is a mapping $F^{N'} : N \cup \mathcal{D} \rightarrow \mathbb{R}^k$, where $|N'| = k$ is the dimensionality of the vector space. A *reference node embedding* of $\mathcal{G}$ based on $N' \subset N$ defines the function $F^{N'}$ as follows. For each $n \in N$, $F^{N'}(n) = (F_1^{N'}(n), \ldots, F_k^{N'}(n))^{\mathbf{T}}$, where $F_i^{N'}(n) = d_{net}(n, n_{r_i})$ for $1 \leq i \leq k$. Objects can be embedded analogously. For each $o \in \mathcal{D}$ located on a node n, $F^{N'}(o) = F^{N'}(n)$. For each $o \in \mathcal{D}$ located on an edge $(n_1, n_2) \in E$, $F^{N'}(o) = (\hat{F}_1^{N'}(o), \ldots, \hat{F}_k^{N'}(o))^{\mathbf{T}}$, where $\hat{F}_i^{N'}(o) = \min\{d_1(o) + F_i^{N'}(n_1), d_2(o) + F_i^{N'}(n_2)\}$.

In Figure 1 a reference node embedding of some objects located on a sample network graph using reference nodes $N' = \langle n_8, n_7 \rangle$ is illustrated.

The reference node embedding has two major advantages. First, if the graph structure remains fixed (which is obviously a realistic assumption) and the embedding of the graph nodes (that do not change) is performed offline in a preprocesing step and is then stored, a re-embedding of moving objects can be done very efficiently. Second, the reference node embedding can be used to compute upper and lower bounds for the network distance. In [14] it is shown that the distance $D(x, y) = \max_{i=1..k} |x_i - y_i|$ in the embedded space lower bounds the distance d_{net} in the native space. In addition, it is shown that the distance function $D^*(x, y) = \min_{i=1\ldots k}(x_i + y_i)$ is an upper bound of d_{net}. In summary, the reference node embedding approach is very suitable to efficiently support similarity queries over both static and dynamic objects in traffic networks.

3.2 The Idea of Hierarchical Network Embedding

Beside these two significant advantages, the reference node embedding proposed in [12, 13] and [14] — in the following called *flat* embedding — has one major shortcoming. The performance gain of the embedding heavily depends on the number of reference nodes used. Though it is shown in [14] that even a low number of reference nodes is sufficient in order to achieve significant performance boosts on small and medium-sized networks, it is also indicated that on large-scale networks, the number of reference nodes necessary to approximate the network distance sufficiently well and to speed-up similarity query processing is considerably large. However, a large set of reference nodes leads to high storage cost because we have to store $O(|N| \cdot |N'|)$ distances for the embedding. In addition, also the computational cost of the embedding and re-embedding process and of the query processor increases with increasing $|N'|$. Especially the increase of query processing cost (due to higher CPU cost to determine the distance between $|N'|$-dimensional points and due to higher I/O cost caused by the fact that higher dimensional points can be indexed less efficiently) is a severe handicap of the flat embedding approach.

Obviously, the reason for this bad scalability of the flat reference node embedding on large networks is the increasing dimensionality of the resulting embedding vectors in the vector space $\mathbb{R}^{|N'|}$. This is somewhat arbitrary because finally only one reference node is taken into account for a distance estimation as D and D^* aggregate over the distances to all reference nodes such that only the "best" reference node is taken. Usually a small subset of the reference nodes suffices for the distance estimation between an object o and any other object in the graph. It is easy to see that the smaller is the distance of a reference node to a particular object o, the better is this reference node for all distance approximations w.r.t. o.

In this paper, we propose a solution to the limited scalability of the flat reference node embedding that is inspired by these considerations. Given an object o there are reference nodes that are more relevant and less relevant for o in N'. So why not use only the relevant reference nodes in N' for the embedding of o? This should decrease the dimensionality of the resulting embedded vectors without downgrading the distance approximations considerably.

3.3 Two-level Network Embedding

A first approach is to use for each object o only the K nearest reference nodes $N'_o \subseteq N'$, where $K << N'$. Obviously, the lower and upper bounding distance approximations D and D^* can still be used to approximate the network distance $d_{net}(x, y)$ between two objects x and y as far as the intersection of the corresponding reference node sets N'_x and N'_y is not empty, i.e. $N'_x \cap N'_y \neq \emptyset$.

However, in large traffic networks with a large reference node set N', it is more likely that this property does not hold for most of the pairs of objects, in particular for those which are not very close to each other. To overcome this problem, we introduce a further embedding level on top of the current embedding. A comprehensive graph $\mathcal{G}' = (N', E', W')$ is built using all reference nodes N' as nodes and all shortest paths

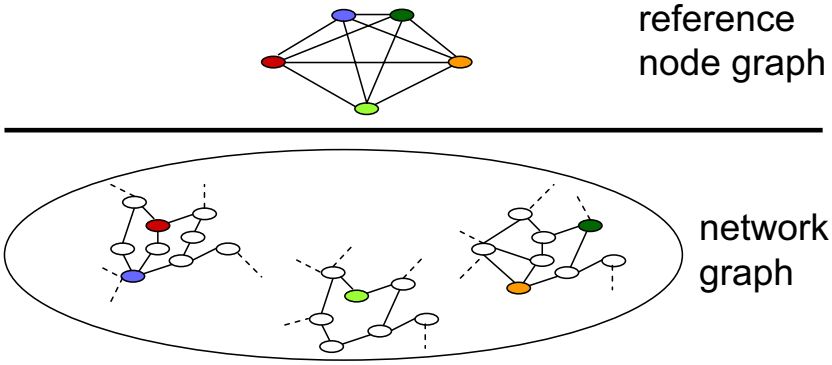


Fig. 2. Schema of a 2-level reference node embedding.

between these nodes in the original graph $\mathcal{G}$ as edges E'. The weights W' are determined analogously. The idea is illustrated in Figure 2.

Formally, let $\mathcal{G} = (N, E, W)$ be the network graph and $N' \subseteq N$ a set of reference nodes (landmarks) with $|N'| \geq K$. For each node or object $o \in N \cup \mathcal{D}$ let $N'_o = \langle r^o_1, \ldots, r^o_K \rangle$, $r^o_j \in N'$ be the set of K local reference nodes relevant for o. The 2-level embedding, or transformation, of the native space $N \cup \mathcal{D}$ into a K-dimensional vector space $\mathbb{R}^k$ is a mapping $\tilde{F}^{N'} : N \cup \mathcal{D} \rightarrow \mathbb{R}^K$ together with a reference node graph $\mathcal{G}' = (N', E', W')$. A *2-level reference node embedding* of $\mathcal{G}$ and $\mathcal{D}$ based on $N' \subset N$ is a pair $(\tilde{F}^{N'}, \mathcal{M}')$ consisting of the mapping function $\tilde{F}^{N'}$ and the reference node matrix $\mathcal{M}'$ that is the weighted adjacent matrix of $\mathcal{G}'$.

The function $\tilde{F}^{N'}$ is defined as follows.

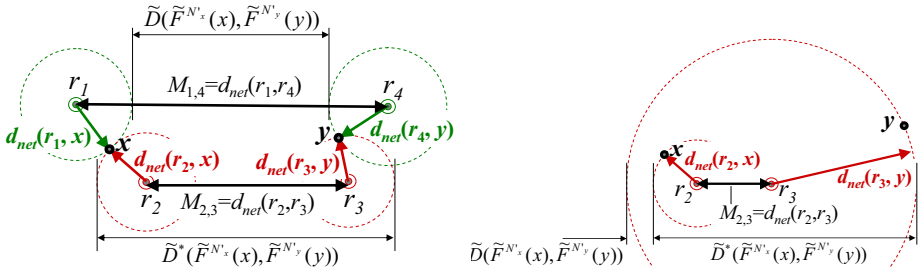
$$\tilde{F}^{N'_o}(o) = \begin{cases} (d_{net}(r^o_1, o), \ldots, d_{net}(r^o_K, o))^{\mathbf{T}} & \text{if } o \in N \text{ is a node} \\ \tilde{F}^{N_n}(n) & \text{if object } o \in \mathcal{D} \text{ is located on } n \in N \\ (S^{N'_o}_1(o), \ldots, S^{N'_o}_k(o))^{\mathbf{T}} & \text{if } o \in \mathcal{D} \text{ is located on } (n_i, n_j) \in E \end{cases}$$

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

Let us note that a re-embedding of moving objects using $\tilde{F}$ is still very efficient as long as we assume that the graph structure remains fixed because then the embedding of the graph nodes performed in a preprocessing step can be stored.

The reference node graph $\mathcal{G}' = (N', E', W')$ is a graph over all reference nodes N', where $E' = \{(n_i, n_j) | n_i, n_j \in N'\}$ is the set of all pairwise connections between the reference nodes in N' and $W'(n_i, n_j) = d_{net}(n_i, n_j)$ is the shortest path between the corresponding reference nodes $n_i, n_j \in N'$ in the original graph $\mathcal{G}$.

Because the set of edges is implicitly defined, we can store and represent this reference node graph by its weighted adjacency matrix which we call the *reference node matrix*. This matrix has the following general form.



(a) case A and D (b) case B and C

Fig. 3. Illustration of the distance approximation derived from a reference node embedding.

$$\mathcal{M}' = \begin{bmatrix} 0 & d_{net}(r_1, r_2) & \dots & d_{net}(r_1, r_k) \\ d_{net}(r_2, r_1) & 0 & \dots & d_{net}(r_2, r_k) \\ \vdots & \vdots & \ddots & \vdots \\ d_{net}(r_k, r_1) & d_{net}(r_k, r_2) & \dots & 0 \end{bmatrix}$$

In summary, the pair $(\tilde{F}^{N'}, \mathcal{M}')$ defines a 2-level reference node embedding of a graph $\mathcal{G}$.

3.4 Distance Approximations

Based on $\tilde{F}^{N'}$ and $\mathcal{M}'$, we can now define a distance function $\tilde{D}$ for objects in $x, y \in \mathcal{D}$ in the embedded space that lower bounds d_{net} as follows.

$$\tilde{D}(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) = \max_{k \in N_x, l \in N_y} \left\{ \begin{array}{ll} M_{i_k,i_l} - \tilde{F}_k^{N_x}(x) - \tilde{F}_l^{N_y}(y) & \text{(case A)} \\ \tilde{F}_k^{N_x}(x) - M_{i_k,i_l} - \tilde{F}_l^{N_y}(y) & \text{(case B)} \\ \tilde{F}_l^{N_y}(y) - M_{i_k,i_l} - \tilde{F}_k^{N_x}(x) & \text{(case C)} \\ 0 & \text{(case D)} \end{array} \right\}$$

where i_p represents the index of the $r_p^n \in N_n$ in $\mathcal{M}'$ and where the following cases appear: case A: $d_{net}(r_i, r_j) > d_{net}(n_a, r_i) + d_{net}(n_b, r_j)$, case B: $d_{net}(n_a, r_i) > d_{net}(r_i, r_j) + d_{net}(n_b, r_j)$, case C: $d_{net}(n_b, r_j) > d_{net}(n_a, r_i) + d_{net}(r_i, r_j)$ and case D otherwise.

Figure 3 illustrates the definition of $\tilde{D}$. On the left hand side, case A ($k = 1$ and $l = 4$, i.e. r_1 and r_4 determine the distance approximation) is visualized. On the right hand side, case B and case C (symmetric) are depicted.

Lemma 1 (Lower bounding property). *Let $(\tilde{F}, \mathcal{M}')$ be a 2-level reference node embedding of nodes and objects of a network $\mathcal{G} = (N, E, W)$ w.r.t. a set of reference nodes N' and local reference node sets N_o for all nodes or objects $o \in N \cup \mathcal{D}$. For each $x, y \in N \cup \mathcal{D}$ the following property holds.*

$$\tilde{D}(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) \leq d_{net}(x, y).$$

Proof. Without loss of generality, let $r_i \in N_x$ und $r_j \in N_y$ be the reference nodes that determine $\tilde{D}$. Since d_{net} is a metric, the following considerations hold.

Case A occurs if $d_{net}(r_i, r_j) > d_{net}(n_a, r_i) + d_{net}(n_b, r_j)$. Then,

$$\begin{aligned}\tilde{D}(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) &= M_{i_i,i_j} - \tilde{F}_i^{N_x}(x) - \tilde{F}_j^{N_y}(y) \\ &= d_{net}(r_i, r_j) - d_{net}(x, r_i) - d_{net}(y, r_j) \\ &\leq d_{net}(r_i, n_a) - d_{net}(n_b, r_j) \\ &= d_{net}(x, r_j) - d_{net}(y, r_j) \\ &\leq d_{net}(x, y)\end{aligned}$$

Case B occurs if $d_{net}(n_a, r_i) > d_{net}(r_i, r_j) + d_{net}(n_b, r_j)$. Then,

$$\begin{aligned}\tilde{D}(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) &= \tilde{F}_i^{N_x}(x) - M_{i_i,i_j} - \tilde{F}_j^{N_y}(y) \\ &= d_{net}(x, r_i) - d_{net}(r_i, r_j) - d_{net}(y, r_j) \\ &= d_{net}(x, r_i) - d_{net}(r_j, r_i) - d_{net}(y, r_j) \\ &\leq d_{net}(x, r_j) - d_{net}(y, r_j) \\ &\leq d_{net}(x, y)\end{aligned}$$

Case C occurs if $d_{net}(n_b, r_j) > d_{net}(n_a, r_i) + d_{net}(r_i, r_j)$. Then,

$$\begin{aligned}\tilde{D}(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) &= \tilde{F}_j^{N_y}(y) - M_{i_i,i_j} - \tilde{F}_j^{N_y}(y) \\ &= d_{net}(y, r_j) - d_{net}(r_i, r_j) - d_{net}(x, r_i) \\ &\leq d_{net}(y, r_i) - d_{net}(x, r_i) \\ &\leq d_{net}(x, y)\end{aligned}$$

Otherwise, in case D, we have

$$\tilde{D}(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) = 0 \leq d_{net}(x, y)$$

Analogously, we can define a distance function $\tilde{D}^*$ for objects in $x, y \in \mathcal{D}$ in the embedded space that upper bounds d_{net} as follwos.

$$\tilde{D}^*(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) = \min_{k \in N_x, l \in N_y} \{M_{i_k,i_l} + \tilde{F}_k^{N_x}(x) + \tilde{F}_l^{N_y}(y)\}$$

where i_p is defined as above. Figure 3 illustrates the definition of $\tilde{D}^*$.

Lemma 2 (Upper bounding property). *Let ($\tilde{F}$,$\mathcal{M}'$) be a 2-level reference node embedding of nodes and objects of a network $\mathcal{G} = (N, E, W)$ w.r.t. a set of reference nodes N' and local reference node sets N_o for all nodes or objects $o \in N \cup \mathcal{D}$. For each $x, y \in N \cup \mathcal{D}$ the following property holds.*

$$\tilde{D}^*(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) \geq d_{net}(x, y).$$

Proof. Let $x, y \in N \cup \mathcal{D}$. Since d_{net} is a metric, for each pair of reference nodes $r_i \in N_x$ und $r_j \in N_y$ the following holds:

$$\begin{aligned}
\tilde{D}^*(\tilde{F}^{N_x}(x), \tilde{F}^{N_y}(y)) &= M_{i_i,i_j} + \tilde{F}_i^{N_x}(x) + \tilde{F}_j^{N_y}(y) \\
&= d_{net}(r_i, r_j) + d_{net}(x, r_i) + d_{net}(y, r_j) \\
&= d_{net}(x, r_i) + d_{net}(r_i, r_j) + d_{net}(r_j, y) \\
&\geq d_{net}(x, r_j) + d_{net}(r_j, y) \\
&\geq d_{net}(x, y)
\end{aligned}$$

Let us note that for directed graphs, d_{net} is no metric distance function (it is not symmetric). However, lower and upper bounds for the network distance on the 2-level embedding can be defined analogously also for directed graphs. Since in this case $d_{net}(x, y)$ is not symmetric we have to distinguish between the two traversal directions ($x \rightarrow y$ and $y \rightarrow x$) for which we have to take the corresponding directed edge weights into account.

3.5 From Two-level to Multi-level Network Embeddings

The proposed 2-level reference node embedding scales very well even for very large graphs as far as the number K of relevant reference nodes for each object is considerably small. We will see this in our experiments (cf. Section 5). However, for very large graph networks that require a high reference node density, K can again be large. In addition, the storage cost for the reference node matrix $\mathcal{M}'$ obviously scale quadratic with the number of global reference nodes N'. In such scenarios, $\mathcal{M}'$ will no longer fit into main memory. This will increase the query processing time dramatically since for determining the distance approximations, we steadily need random access to the elements in $\mathcal{M}'$.

To solve this problem, we propose to introduce further embedding levels, i.e. to generalize the 2-level reference node embedding to a multi-level reference node embedding. Such a multi-level embedding can be constructed bottom-up starting with a 2-level embedding. The reference node set is partitioned at each level. Each of these partitions is assigned to one of the objects/nodes in the network as the corresponding relevant reference node set. The reference node partitions may overlap and neighboring nodes/objects should get nearly the same reference node partition assigned. For each partition, a complete reference node graph is constructed on the second embedding level. Thereby, the size of each partition should be chosen such that the reference nodes on each level are completely connected (i.e. each reference node is reachable from each other) when combining all reference node graphs on a level. Furthermore, the reference node matrices of the corresponding reference node graphs should fit into a memory page. From the resulting reference node graph on embedding level i a predefined number of nodes is selected to form an embedding level $i + 1$ analogously. This procedure is iterated until only one "graph" remains that is complete and fits into main memory. The idea is illustrated in Figure 4.

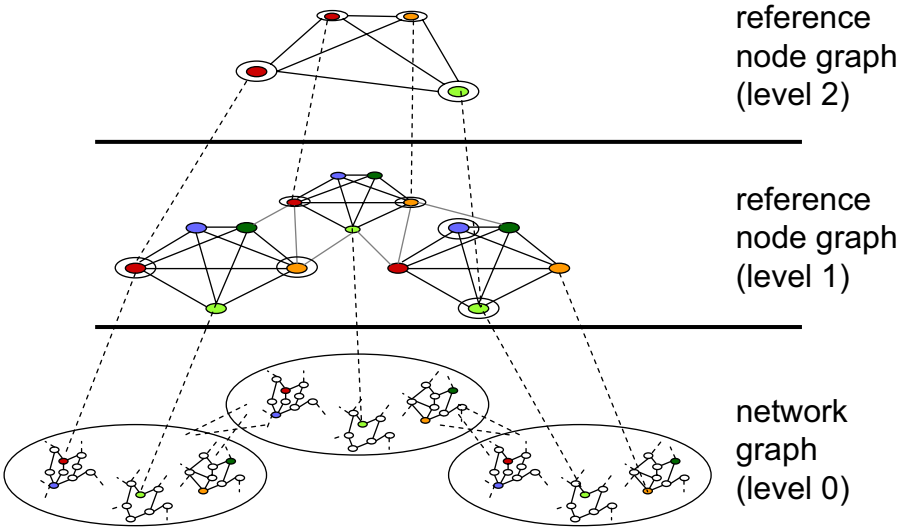


Fig. 4. Schema of a multi-level reference node embedding.

3.6 Choosing the Reference Node Set

It is easy to see that the choice of the reference node set affects the quality of the distance approximations. The problem of how to choose the reference nodes is two-fold. First, the global set N' has to be chosen adequately, and second, for each node or object $o \in N \cup \mathcal{D}$, the local set of relevant reference nodes N'_o needs to be selected, too. Obviously, the choice of the global set N' affects the possibilities for the selection of the local sets N'_o.

For a flat embedding, $D(x, y) = d_{net}(x, y)$ for two objects $x, y \in \mathcal{D}$, i.e. the approximation error is zero, if there is at least one $r_i \in N'$ such that either $x \in \mathcal{P}_{best}(r_i, y)$ or $y \in \mathcal{P}_{best}(r_i, x)$, where $\mathcal{P}_{best}(a, b)$ denotes the shortest path between nodes/objects a and b. On the other hand, the approximation obtained from reference node r_i is very coarse if r_i is located such that $d_{net}(x, r_i) \approx d_{net}(y, r_i)$ and $r_i \in \mathcal{P}_{best}(x, y)$. On the other hand, $D^*(x, y) = d_{net}(x, y)$ for two objects $x, y \in \mathcal{D}$, i.e. the approximation error is zero, if there is at least one $r_i \in N'$ such that $r_i \in \mathcal{P}_{best}(x, y)$. The more disconnected a reference node r_i is from $\mathcal{P}_{best}(x, y)$, the coarser is the approximation obtained from this reference node. The same considerations hold true for a 2-level or even for a multi-level reference node embedding.

Intuitively, the probability that these conditions for accurate distance approximations are fullfilled is higher if the reference nodes are close to the objects. Thus, if the distribution of the objects in the network is unknown, the set of global reference nodes N' should be evenly distributed over the network because then, the probability that all objects have at least one reference node in their local vincinity is maximized. In addition, for each node or object $o \in N \cup \mathcal{D}$, the set of relevant reference nodes N'_o should be selected as the K-nearest reference nodes of o from N'. This further ensures that the reference nodes N'_o are in proximity of o.

On the other hand, if information about the object distribution, the characteristics of object movement, and/or the distribution of query locations is known, the set of local relevant reference nodes N'_o for all nodes or objects $o \in N \cup \mathcal{D}$ could be selected individually. Ideally, hot spots, i.e. nodes that are often part of shortest paths during query execution, should be chosen as reference nodes. Since the set N'_o of reference nodes relevant for node/object o can be dynamically adjusted rather easily, we can even learn the location of hot spots by monitoring for each node how often it is visited during a shortest path computation.

3.7 Efficient Shortest-path Computation

Analogously to the flat embedding, our hierarchical reference node embedding can be successfully applied as heuristics for the A*-search algorithm to compute the true network distance. The A*-search method is a special case of a best-first search algorithm using heuristics. In contrast to the Dijkstra algorithm whose search is only backward-oriented (blind search), the A*-search method is an informed search method, i.e. it also looks in the forward direction using a lower bounding network distance approximation, e.g. the Euclidean distance. Here, we propose to use the distance function $\tilde{D}$ of the vector space resulting from our multi-level K-closest reference node embedding as estimator function. In addition, we can use the upper bounding distance estimation $\tilde{D}^*$ in order to identify the branches of the search tree that do not need to be expanded. Since these branches do not need to be considered throughout the remaining search steps, we do not need to maintain them which reduces the memory cost.

4 Multi-step Query Processing

The upper and lower bounding distance estimations introduced above can be used in a filter step as well as for speeding-up the refinement step using the modified A* algorithm. In the following, we present the multi-step DRQ and kNNQ using our embedding function $F^{N'}$ implementing a multi-level K-closest 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. The re-embedding for dynamic objects can be computed rather efficiently on the fly (cf. Section 3).

The DRQ over the embedded objects and nodes can directly prune all objects for which the distance approximation $\tilde{D}$ is greater than ε as true drops without refining them. All objects are added to the result list if the distance estimation $\tilde{D}^*$ is lower or equal to ε. Only the remaining candidates need to be refined.

For the kNNQ we use the algorithm proposed in [18] which is shown to be optimal w.r.t. the number of candidates that are refined. The algorithm is illustrated in Figure 5. It uses a ranking of the objects in ascending order of their lower bounding filter distance $\tilde{D}$ and performs 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.

```
kNNQ(q,k,G)
  SortedList results,candidates;
  initialize ranking := RQ(q,D);
  candidates←first k objects from ranking;
  d_min = k^th smallest D(F^N'(q), F^N'(o)) of o ∈candidates;
  d_max = k^th smallest D*(F^N'(q), F^N'(o)) of o ∈candidates;
  d_f_next = D(F^N'(q), F^N'(o)) of o=ranking.top_element;
  do {
    update d_min, d_max, and d_f_next;
    if d_min ≥ d_f_next then
      candidates.add(ranking.top_element);
      update d_min, d_max, and d_f_next;
    for all c ∈ candidates do
      if D*(F^N'(q), F^N'(c)) < d_min then add c to result;
      if D(F^N'(q), F^N'(c)) > d_max then prune c;
    if |results|+|candidates| > k ∨ d_f_next ≤ d_max then
      for all c ∈ candidates with D(F^N'(q), F^N'(c)) ≤ d_min
                                  ∧ d_max ≤ D*(F^N'(q), F^N'(c)) do
        if d_net(q,c) ≤ d_k-nn(q, result) then add c to result;
    else add all remaining c ∈ candidates to result;
  } while (d_f_next ≤ d_max ∨ |candidates| > 0)
  return result;
```

Fig. 5. The kNNQ algorithm.

5 Experimental Evaluation

Due to space limitations, we focus on a two-level embedding in our experiments. We used real road networks of San Joaquin County ("TG", 18,300 nodes) and San Francisco ("SA", 175,000 nodes). The network objects were simulated through randomized samples of the graph nodes. The graph was stored on disk implementing the approach proposed in [8] using R*-trees with a block size of 8 KB and an average storage load of 70% each. The R*-trees are used to manage the nodes, the edges and the street segments in form of polylines. An embedding vector is a further attribute of a node. 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 by REF, flat embeddings by 1RNE and two-level embeddings by 2RNE. All experiments were performed on a workstation featuring a 1.8 GHz CPU, 2GB RAM, a random disk with 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.

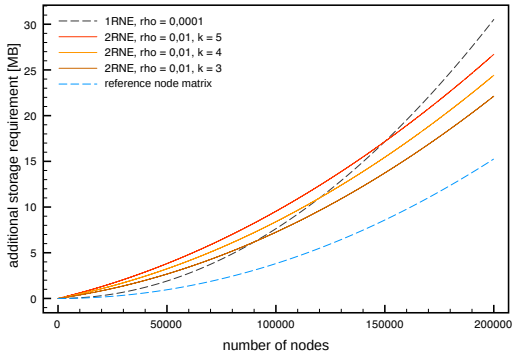


Fig. 6. Size of the embedding, w.r.t. size of the network graph.

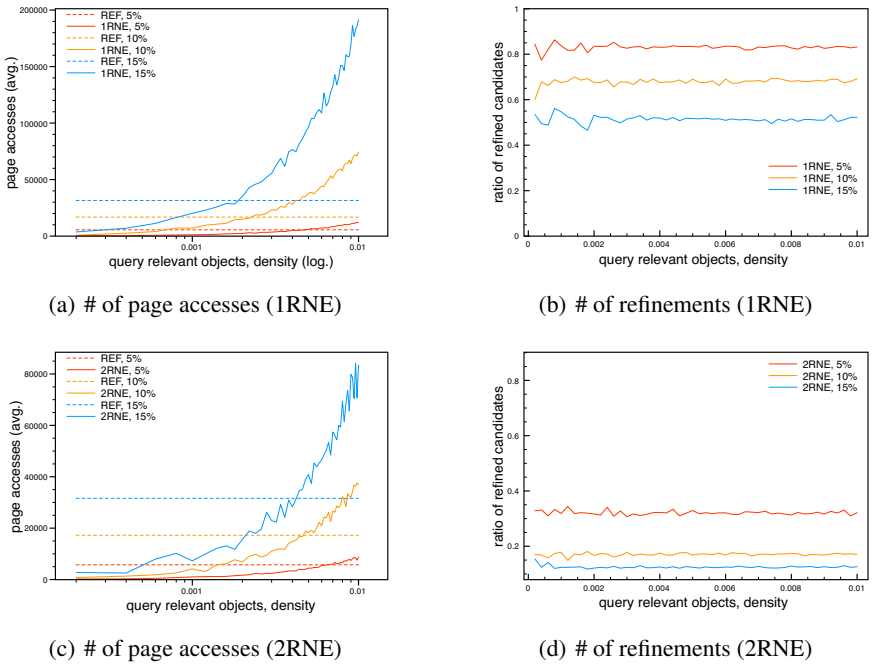
5.1 Storage Requirements

Figure 6 shows the storage requirements of different embeddings. We compared a flat embedding (1RNE) with an *object density* of *rho* = $0,0001$ (*rho* = # objects / # graph nodes) with several two-level embeddings (2RNE) using different numbers K of *relevant reference nodes*[1] per node and object. For each 2RNE we assumed a considerably higher object density of *rho* = $0,01$. In addition, the size of the reference node matrix $\mathcal{M}'$ is depicted. In the following, M denotes the size of the reference node matrix $\mathcal{M}'$, i.e. about $\sqrt{M}$ (1^{st}-level) reference nodes are used for $\mathcal{M}'$. It can be observed, that using a 2RNE we can use approximately two orders of magnitude more reference nodes compared to a 1RNE with quite similar storage cost. Obviously, using more global reference nodes increases the quality of the distance approximations, and, thus boosts the overall performance.

5.2 Multi-Step Query Processing

In this experiment, we assumed a capacity of 80 byte per embedded node and object of the network graph. We used $K = 3$ relevant reference nodes per node and object resulting in an overall number of 400 and 700 reference nodes for TG and SF, respectively. The reference node distance matrix $\mathcal{M}'$ thus required 0.61 MB (TG) and 1,88 MB (SF) RAM, respectively. Because of its small size, the distance matrix $\mathcal{M}'$ was kept in main memory in all experiments. The results of distance range query processing on the SF dataset are depicted in Figure 7. The most important advantage of the 2RNE over the 1RNE approach is that it can use significantly more reference nodes which increases the quality of the filter distances. Especially for less selective queries, the filter selectivity is significantly better than using a 1RNE with comparable storage requirements. The scalability of the 2RNE approach w.r.t. the object density is linear similar to the 1RNE approach. The results on the TG dataset are similar (not shown due to space limitations). In summary, the results show that the 2RNE approach is superior to 1RNE especially

[1] The number K of relevant reference nodes corresponds to the number of reference nodes assigned to each graph node on each embedding level.



(a) # of page accesses (1RNE)

(b) # of refinements (1RNE)

(c) # of page accesses (2RNE)

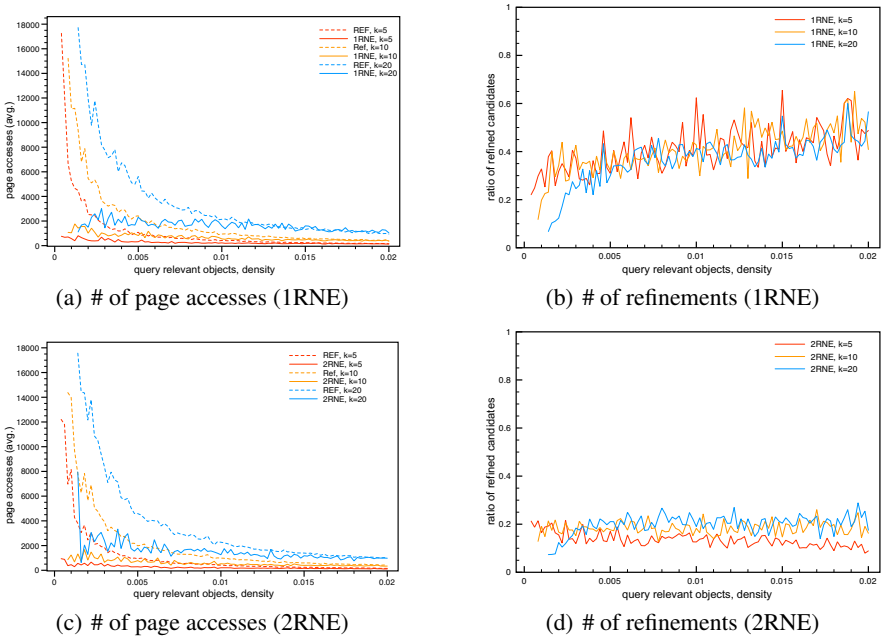
(d) # of refinements (2RNE)

Fig. 7. Performance of DRQ w.r.t. the object density *rho* on SF dataset.

on large graphs with comparable storage requirements. The results of kNNQ processing on the TG dataset are depicted in Figure 8. On the SF dataset we made similar observations (not shown due to space limitations). Here, especially for high object densities, the 2RNE approach outperforms the 1RNE approach on large datasets. Again, the higher number of reference nodes that can be used in the 2RNE approach yields a significantly better distance approximation.

5.3 Shortest Path Algorithm

In this experiment, we concentrate on the cost required for the refinement step, i.e. the cost of the exact distance computation. The benefits of our novel distance approximations for computing the shortest path can be observed in Figure 9. A sample shortest path (marked in blue) is computed by four different methods. For each method, the corresponding search space containing all visited edges is marked in orange. While the Dijkstra algorithm (cf. Figure 9(a)) needs to access nearly the complete displayed part of the graph, the A* algorithm using the Euclidean distance as lower bounding distance estimation (cf. Figure 9(b)) requires a considerably reduced search space. The novel A* algorithm with upper and lower bounding distance estimations derived from a flat reference node embedding with $|N'| = 50$ reference nodes (cf. Figure 9(c)) further reduces the search space. Finally, our 2-level reference node embedding with $|N'| = 100$



(a) # of page accesses (1RNE)

(b) # of refinements (1RNE)

(c) # of page accesses (2RNE)

(d) # of refinements (2RNE)

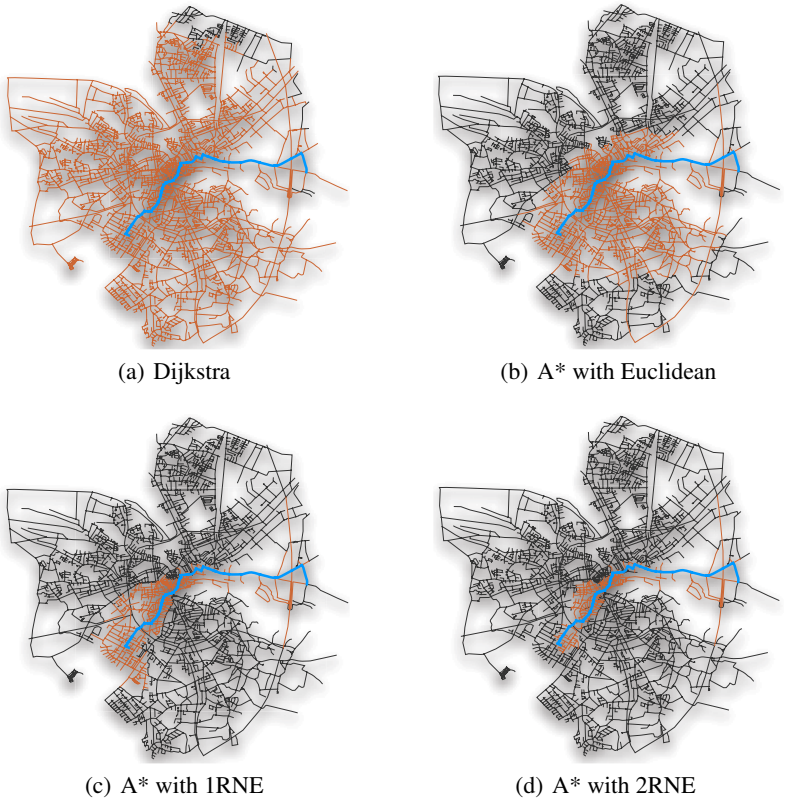
Fig. 8. Performance of kNNQ w.r.t. the object density *rho* on TG dataset.

global reference nodes and $K = 5$ relevant reference nodes per object (cf. Figure 9(d)) requires the smallest search space of all competitors.

Figures 10(a) and 10(b) show the average number of disk page accesses required for one distance computation between two objects (nodes) w.r.t. the size of the reference node matrix $\mathcal{M}'$ and the number K of reference nodes used for the two-level embedding. As can be observed, our novel shortest path algorithm again significantly outperforms the A* search using the Euclidean distance and Dijkstra (not shown for clarity reasons). We also observed, that increasing the number K of relevant reference nodes per object does not significantly increase the quality of the distance approximation (not shown due to space limitations). Our experiments suggest that $K = 5$ is a reasonable choice despite it is a rather small value. As a consequence, the storage requirements for each object are rather low. In turn, this allows us to use a higher number of global reference nodes.

5.4 Comparison with other Approaches

Finally, we compare the performance of our approach to that of state-of-the art approaches. We chose the distance signature (DS) approach [11] as comparison partner because it outperforms other methods such as the network voronoi diagram [15]. The DS method was parameterized as described in [11]. For the comparison, we computed a two-level (2RNE) embedding for $M = 256$ and $K = 5$. For an object density $rho = 0.01$, the 2RNE embedding occupies half of the space required by DS. Please



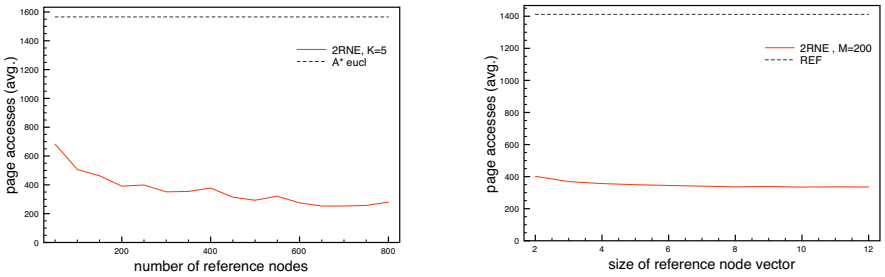
(a) Dijkstra (b) A* with Euclidean

(c) A* with 1RNE (d) A* with 2RNE

Fig. 9. Search space (orange) for computing a sample shortest path (blue).

note that the object density linearly influences the memory footprint of our technique, in contrast to the DS approach where the relationship between object density and memory consumption is quadratic. The DRQ experiments in Figure 11 show that the signature approach is significantly outperformed by our approach. The two-level embedding is able to outmatch DS although it occupies significant less memory, i.e. needs far less precomputed distance information.

In summary, our experimental evaluation empirically showed the following facts: First, the integration of our novel upper and lower bounding distance approximations into the A* algorithm is superior to state-of-the-art methods for shortest path computation. Second, our novel two-level (or even multi-level) embedding outperforms the flat embedding on large graphs because it allows an even more accurate lower and upper bounding distance approximation.



(a) # of page accesses w.r.t. $|\mathcal{M}'|$). (b) # of page accesses w.r.t. K.

Fig. 10. Performance evaluation of the shortest path algorithm.

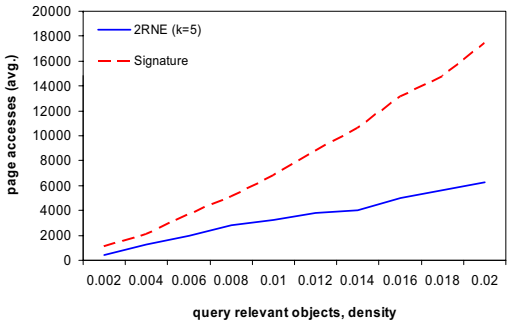


Fig. 11. RNE vs. Signature, w.r.t. the object density *rho*.

6 Conclusions

We proposed a hierarchical graph embedding of very large networks that is suitable for static and dynamic objects. From the embedding, we derived accurate upper and lower bounds for the network distance that can be used to implement a filter/refinement architecture for similarity search in large traffic networks. In addition, our embedding allows an acceleration of the refinement step by applying an informed A*-search using our novel distance approximations. Our experiments show that our novel approach outperforms a simple flat embedding and other existing competitors in terms of pruning power in the filter step and overall performance. Furthermore, it turned out that our informed search in the refinement step is much more efficient than comparable approaches due to a dramatically reduced search space.

References

1. Dijkstra, E.W.: "A Note on Two Problems in Connection with Graphs". Numerische Mathematik **1** (1959) 269–271
2. Corman, T.H., Leiserson, C.E., Riverst, R.L.: "Introduction to Algorithms". MIT Press (1990)

3. Kung, R., Hanson, E., Ioannidis, Y., Sellis, T., Shapiro, L., Stonebraker, M.: "Heuristic Search in Data Base Systems". Expert Database Systems (1986)
4. Agrawal, R., Dar, S., Jagadish, H.: "Direct Transitive Closure Algorithms: Design and Performance Evaluation". TODS **15**(3) (1990)
5. Ioannidis, Y., Ramakrishnan, R., Winger, L.: "Transitive Closure Algorithms Based on Graph Traversal". TODS **18**(3) (1993)
6. Jung, S., Pramanik, S.: "HiTi Graph Model of Topographical Roadmaps in Navigation Systems". In: Proc. Int. Conf. on Data Engineering (ICDE'96). (1996)
7. Köhler, E., Möhring, R.H., Schilling, H.: "Acceleration of Shortest Path and Constrained Shortest Path Computation". In: Proc. Int. WS Efficient and Experimental Algorithms (WEA'05). (2005)
8. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: "Query Processing in Spatial Network Databases". In: Proc. Int. Conf. on Very Large Databases (VLDB'03). (2003)
9. Shahabi, C., Kolahdouzan, M., Sharifzadeh, M.: "A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases". Geoinformatica **7**(3) (2003) 255–273
10. Linial, N., London, E., Rabinovich, Y.: "The geometry of graphs and some of its algorithmic applications". In: Proc. IEEE Symp. Foundations of Computer Science. (1994)
11. Hu, H., Lee, D.L., Lee, V.C.S.: "Distance Indexing on Road Networks". In: Proc. Int. Conf. on Very Large Databases (VLDB'06). (2006)
12. Goldberg, A.V., Werneck, R.F.: "Computing Point-to-Point Shortest Paths from External Memory". In: Proc. of the 7th WS on Algorithm Engineering and Experiments (ALENEX), SIAM. (2005)
13. Goldberg, A.V., Kaplan, H., Werneck, R.F.: "Reach for A*: Efficient point-to-point shortest path algorithms". In: Proc. of the 8th WS on Algorithm Engineering and Experiments (ALENEX), SIAM. (2006)
14. Kriegel, H.P., Kröger, P., Kunath, P., Renz, M., Schmidt, T.: "Proximity Queries in Large Traffic Networks". In: Proc. 15th Int. Symposium on Advances in Geographic Information Systeme (ACM GIS'07), Seattle, WA. (2007)
15. Kolahdouzan, M., Shahabi, C.: Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases". In: Proc. Int. Conf. on Very Large Databases (VLDB'04). (2004)
16. Kolahdouzan, M., Shahabi, C.: "Continuous K-Nearest Neighbor Queries in Spatial Network Databases". In: In Proc. of STDBM, 2004. (2004)
17. Cho, H.J., Chung, C.W.: An efficient and scalable approach to cnn queries in a road network. In: Proc. Int. Conf. on Very Large Databases (VLDB'05). (2005)
18. Kriegel, H.P., Kröger, P., Kunath, P., Renz, M.: "Generalizing the Optimality of Multi-Step k-Nearest Neighbor Query Processing". In: Proc. 10th Int. Symp. on Spatial and Temporal Databases (SSTD'07), Boston, MA. (2007)