In Proc. 24th International Conference on Data Engineering (ICDE'08), Cancun, Mexico, 2008.

Efficient Query Processing in Large Traffic Networks

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

Ludwig-Maximilians-Universität München, Oettingenstr. 67, 80538 Munich, Germany
Email: {kriegel,kroegerp,kunath,renz,schmidtti}@dbs.ifi.lmu.de

***Abstract*— We present an original graph embedding to speed-up distance-range and k-nearest neighbor queries on static and/or dynamic objects located on a (weighted) graph. Our method is used to compute a lower and upper bounding filter distance which approximates the true shortest path distance significantly better than traditional filters. In addition, we discuss how the computation of the exact shortest path distance in the refinement step can be boosted by using the embedded graph.**

I. Introduction

Efficient support of proximity queries in large traffic networks are required 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 distance between objects located on the network is usually measured by means of the shortest path distance which is commonly computed by the Dijkstra algorithm.

In this paper, we propose a novel multi-step query processor for large graph networks based on a very simple but effective network graph embedding method. A two-step query 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 by computing the true network distance. 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. Furthermore, we show how the lower bounding distance can be used as a very effective heuristic to guide an informed A*-search for the refinement step.

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 $W : E \rightarrow \mathrm{R}^+$ 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. In the example network graph shown in Figure 1, 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:

1) distance range query (DRQ): Given a query object q located on $\mathcal{G}$ and a distance threshold $\varepsilon \in \mathrm{R}^+$, a *distance range query* (DRQ) returns the set $DRQ(q, \varepsilon) = \{o \in \mathcal{D} \,|\, d_{net}(q, o) \leq \varepsilon\}$.

2) k-nearest neighbor query (kNNQ): Given a query object q located on $\mathcal{G}$ and a number $k \in \mathrm{N}^+$, a *k-nearest neighbor query* (kNNQ) returns the set $NNQ(q, k)$ containing k objects such that $\forall o \in NNQ(q, k), \hat{o} \in \mathcal{D} \backslash NNQ(q, k) : d_{net}(q, o) \leq d_{net}(q, \hat{o})$.

II. 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], [5] suffer from increasing storage costs. In [6] 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 [7] the Euclidean distance between graph nodes/objects is used as a lower bounding filter in order to guide an incremental network expansion for the refinement step. In [8] one of the graph embedding technique from [9] is applied in order to estimate the network distance between two nodes. As 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 [10] *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.

The drawback of this proposal is that proximity queries on moving objects that frequently change their positions are not well supported. In [11] 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. In case of dense network graphs, the computation of the kNN would have a poor performance due to the resulting large size of the cells.

III. Network Graph Embedding

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 a Lipschitz embedding using singleton reference sets called *reference nodes*.

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, d_{net}) into a k-dimensional vector space (R^k, D) is a mapping $F^{N'} : N \cup \mathcal{D} \rightarrow \mathrm{R}^k$, where $|N'| = k$ is the dimensionality of the vector space and D is the L_∞-norm in R^k, i.e. $D(x, y) = \max_{i=1..k} |x_i - y_i|$, where $x, y \in \mathrm{R}^k$ are two points of the vector space (R^k, D). 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 $1 \leq i \leq k$.

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

An example demonstrating the embedding of the objects located on a network graph using reference nodes $N' = \langle n_8, n_7 \rangle$ is depicted in Figure 1.

For the embedding of a network graph we have to compute for each node and each object the shortest paths to all reference nodes. This computation can be done by starting at each reference node the Dijkstra shortest path algorithm and storing the shortest distance at each visited node. This operation has to be performed only once in a preprocessing step.

Dynamic objects can be handled as follows. 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 can be very efficiently computed assuming that $F^{N'}(n_1)$ and $F^{N'}(n_2)$ is given. For this reason, our embedding is also very suitable for dynamic objects.

The reference node embedding can be used to compute upper and lower bounds for the network distance.

Lemma 1 (lower bounding property):
Let $\mathcal{G} = (N, E, W)$ and $F^{N'}$ be the reference node embedding of $\mathcal{G}$ w.r.t. $N' \subset N$. For any two nodes $n_i, n_j \in N \cup \mathcal{D}$, $D(F^{N'}(n_i), F^{N'}(n_j)) \leq d_{net}(n_i, n_j)$.

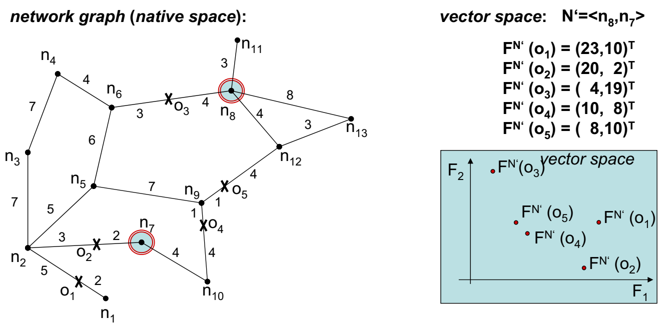

Fig. 1. Network graph embedding.

Proof: Let $N' = \langle n_{r_1}, .., n_{r_k} \rangle$. Since the network distance is transitive, the following statements hold: $D(F^{N'}(n_i), F^{N'}(n_j)) = \max_{l=1,\ldots,k} |F_l^{N'}(n_i) - F_l^{N'}(n_j)| = \max_{l=1,\ldots,k} |d_{net}(n_i, n_{r_l})) - d_{net}(n_j, n_{r_l}))| \leq d_{net}(n_i, n_j)$ ■

The embedded space can also be used to define a progressive approximation of the network distance. The distance function $D^*(x, y) = \min_{i=1...k}(x_i + y_i)$ is an upper bound of d_{net}, formally

Lemma 2 (upper bounding property):
Let $\mathcal{G} = (N, E, W)$ and $F^{N'}$ be the reference node embedding of $\mathcal{G}$ w.r.t. $N' \subset N$. For any two nodes $n_i, n_j \in N \cup \mathcal{D}$, $D^*(F^{N'}(n_i), F^{N'}(n_j)) \geq d_{net}(n_i, n_j)$.

Proof: Let $N' = \langle n_{r_1}, .., n_{r_k} \rangle$. Due to the transitivity of the network distance, the following statements hold: $D^*(F^{N'}(n_i), F^{N'}(n_j)) = \min_{l=1,\ldots,k} (F_l^{N'}(n_i) + F_l^{N'}(n_j)) = \min_{l=1,\ldots,k} (d_{net}(n_i, n_{r_l})) + d_{net}(n_j, n_{r_l}))) \geq d_{net}(n_i, n_j)$ ■

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 (cf. Section V).

IV. Efficient Shortest-path Computation

In this section, we show how the available information of our 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 forward oriented search heuristics. Here, we propose to use the lower bounding distance estimation function D for the search heuristics. In addition, we can use the upper bounding distance estimation D^* in order to identify the branches of the search tree that does not need to be further expanded. The modified A*-search algorithm using our novel distance approximations based on reference node embedding is depicted in Figure 2. The function $C(n_s, n_i)$ denotes the length of the currently determined shortest-path from node n_s to node n_i.

V. 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 algorithm ShortestPath (cf.

```
ShortestPath($n_s$,$n_d$,$\mathcal{G}$)
  $d_{max} = \infty$;
  $nodeSet := \{n_s\}$);
  **for each** $n \in N$ **do**
    $C(n_s, n) = \infty$; $C(n, n) = 0$; $path(n_s, n) := \emptyset$;
  **while** $|nodeSet| > 0$ **do**
    choose $n$ from $nodeSet$
      with minimum $C(n_s, n) + D(F^{N'}(n), F^{N'}(n_d))$;
    **if** $d_{max} > C(n_s, n) + D^*(F^{N'}(n), F^{N'}(n_d))$ **then**
      $d_{max} = C(n_s, n) + D^*(F^{N'}(n), F^{N'}(n_d))$;
    **if** $n = n_d$ **then**
      report $path(n_s, n)$,$C(n_s, n)$ and terminate;
    **else**
      **for each** $n_i \in n.adjacencyList$ **do**
        **if** $C(n_s, n_i) > C(n_s, n) + W(\langle n, n_i \rangle)$ **then**
          $C(n_s, n_i) := C(n_s, n) + W(\langle n, n_i \rangle)$
          $path(n_s, n_i) := path(n_s, n) + \langle n, n_i \rangle$;
          **if** $C(n_s, n_i) + D(F^{N'}(n_i), F^{N'}(n_d)) < d_{max}$
            and $n_i \notin nodeSet$ **then**
              push $n_i$ to $nodeSet$;
```

Fig. 2. Shortest-path algorithm.

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

Fig. 3. DRQ algorithm.

Figure 2). In the following, we present the *multi-step distance range query* (DRQ) and *multi-step k-nearest neighbor query* (kNNQ) using our embedding function $F^{N'}$ implementing a reference node embedding.

The DRQ over the embedded objects and nodes can directly prune all objects for which the distance approximation D is greater than ε as true drops without refining them. All objects are added to the result list if the distance estimation D^* is lower or equal to ε. Only the remaining candidates need to be refined. The pseudocode for a DRQ is given in Figure 3.

For the kNNQ we use the algorithm proposed in [12] which is shown to be optimal w.r.t. the number of candidates that are refined. The algorithm is depicted in Figure 4. It uses a ranking of the objects in ascending order of their lower bounding filter distance 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.

```
$k$NNQ($q$,$k$,$\mathcal{G}$)
  SortedList *results*,*candidates*;
  initialize ranking := $RQ(q,\mathcal{D})$;
  *candidates*←first $k$ objects from *ranking*;
  $d_{min} = k^{th}$ smallest $D(F^{N'}(q), F^{N'}(o))$ of $o \in$*candidates*;
  $d_{max} = k^{th}$ smallest $D^*(F^{N'}(q), F^{N'}(o))$ of $o \in$*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** $d_{min} \geq d_{f\_next}$ **then**
      *candidates*.add(*ranking*.top_element);
      update $d_{min}$, $d_{max}$, and $d_{f\_next}$;
    **for all** $c \in$ *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*$| > k \vee d_{f\_next} \leq d_{max}$ **then**
      **for all** $c \in$ *candidates* with $D(F^{N'}(q), F^{N'}(c)) \leq d_{min}$
                                        $\wedge\, d_{max} \leq D^*(F^{N'}(q), F^{N'}(c))$ **do**
        **if** $d_{net}(q, c) \leq d_{k-nn}(q, result)$ **then** add $c$ to *result*;
    **else** add all remaining $c \in$ *candidates* to *result*;
  } **while** ($d_{f\_next} \leq d_{max} \vee |$*candidates*$| > 0$)
  **return** *result*;
```

Fig. 4. kNNQ algorithm.

References

[1] E. W. Dijkstra, ""A Note on Two Problems in Connection with Graphs"," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[2] T. H. Corman, C. E. Leiserson, and R. L. Riverst, *"Introduction to Algorithms"*. MIT Press, 1990.

[3] R. Kung, E. Hanson, Y. Ioannidis, T. Sellis, L. Shapiro, and M. Stonebraker, ""Heuristic Search in Data Base Systems"," *Expert Database Systems*, 1986.

[4] R. Agrawal, S. Dar, and H. Jagadish, ""Direct Transitive Closure Algorithms: Design and Performance Evaluation"," *TODS*, vol. 15, no. 3, 1990.

[5] S. Jung and S. Pramanik, ""HiTi Graph Model of Topographical Roadmaps in Navigation Systems"," in *Proc. Int. Conf. on Data Engineering (ICDE'96)*, 1996.

[6] 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.

[7] 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.

[8] C. Shahabi, M. Kolahdouzan, and M. Sharifzadeh, ""A Road Network Embedding Technique for k-Nearest Neighbor Search in Moving Object Databases"," *Geoinformatica*, vol. 7, no. 3, pp. 255–273, 2003.

[9] 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.

[10] 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.

[11] 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.

[12] H.-P. Kriegel, P. Kröger, P. Kunath, and M. Renz, ""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.