Memory-Efficient A*-Search using Sparse Embeddings

Franz Graf Hans-Peter Kriegel Matthias Renz Matthias Schubert
Institute for Informatics
Oettingenstr. 67
D-80538, Munich, Germany
{graf, kriegel, renz, schubert}@dbs.ifi.lmu.de

ABSTRACT

When searching for optimal paths in a network, algorithms like A*-search need an approximation of the minimal costs between the current node and a target node. A reference node embedding is a universal method for making such an approximation working for any type of positive edge weights. A drawback of the approach is that it is necessary to store the shortest distance to each landmark node for each considered attribute. Thus, the memory consumption of the embedding is linearly increasing with the number of attributes and landmarks. Thus, an embedded graph might not be well-suited for handheld devices and may significantly increase the loading cost. In this paper, we propose methods for significantly decreasing the memory consumption of embedded graphs and examine the impact of the landmark selection. Furthermore, we propose to limit the number of embedded nodes in the network and propose an algorithm for shortest path computation working on networks for which only a portion of nodes store an embedding. Finally, we propose a heuristic algorithm for finding a suitable subset of nodes that should be embedded in order to guarantee reasonable computation times. Our experimental evaluation examines the trade-off between embedding memory and processing times on two real-world data sets.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory, Algorithms, Performance

Keywords

GIS, embedding, graph, search, performance, query processing

1. INTRODUCTION

Organizing data in a network is quite common in various application areas like social networks, protein interaction networks, navigation graphs or road networks. In all of these application areas the network distance, reflecting the distance between two network nodes (or objects on the network graph), plays an important role, e.g. for navigation, route planning, spatial queries in the context of location based services, etc. The network distance (which we simply call distance in the remainder) between two nodes v_i and v_j denotes the cost (length) of the shortest path between v_i and v_j and can be determined by aggregating the number of edges or edge weights when traversing on the shortest path between v_i and v_j. As the computation of the network distance is too expensive for many algorithms, many applications apply approximations for the network distance that can be computed more efficiently. Often, network distance approximations that lower bound the network distance are required, e.g. to guarantee no false dismissals when aiming at short distances which is usually the case for most applications. In the following, we call an approximation of the network distance between two nodes *optimistic approximation*, if it is smaller than or equal to the corresponding network distance.

The most prominent algorithm employing an optimistic approximation between two nodes is A^*-search which is used to compute the shortest path between these nodes. In contrast to the well-known Dijkstra algorithm which takes the current cost (i.e. the cost from the start node to the current location when traversing the network graph) into account, A^*-search additionally looks ahead (towards the target) and adds an optimistic approximation from the current location to the target. As a result, A^*-search can direct the graph traversal towards the target node and thus, it significantly reduces the number of accessed nodes during the search. Let us note that the performance of A^*-search predominantly depends on the quality (accuracy) of the employed approximation. Let us note that Dijkstra's algorithm is a special case of the A^*-search when using zero as optimistic approximation.

Methods for generating optimistic approximations are usually strongly connected to a specific type of edge weight. For example, when weighting the edges by their length in a road network, a well-established approximation method is to compute the Euclidian distance between the nodes. Depending on the topology of the network, this heuristic is sometimes very efficient. However, when considering the map of a mountain area, the direct path might be blocked by vertical drop, rivers or other obstacles and the approximation distance is often much smaller than the true distance. Another drawback of this approach is that it is not transferable to any other type of cost. For example, when considering the number of traffic lights as a cost function, the correlation of path length and the number of traffic lights is rather small. Though there exist various other specialized heuristics for distance approximation, it is an important requirement for an approximation to work on arbitrary types of edge weights.

ACM SIGSPATIAL '10 San Jose, California USA

A method allowing this computation is the reference node embedding proposed in [1, 22, 32, 26, 13, 15]. The idea of the embedding is to select a set of reference nodes in the network and compute the distance from each node of the network to each of the reference nodes. After embedding the network with the distances to the reference nodes, it is possible to make optimistic approximations. The positive difference between the distances of each compared node to a common reference node yields a positive approximation. In the worst case, the reference node has the same network distance to both nodes and thus, the approximation is zero. However, in the best case one of the nodes is contained on a shortest path between the other node and the reference node. In this case, the approximation even exactly predicts the network distance (cf Figure 3). To increase the approximation quality it is possible to combine approximations w.r.t. various reference nodes by taking the maximum of each approximation. Since each approximation is guaranteed to be minimal, taking their maximum still yields an optimistic approximation. Thus, the quality of the embedding can be considerably increased by adding additional reference nodes.

The major drawback of the reference node embedding is the additional memory required by the embedding. The embedding needs to store a distance to each reference node for each node. Additionally, modern systems often consider several edge attributes. Though it is possible to approximate any type of positive edge weight by the embedding, each type of weight has to be approximated by its own embedding information. Since the number of reference nodes required for a sufficiently good approximation increases with the size of the graph, the memory consumption of the embedding increases significantly for larger graphs disqualifying the applicability of the graph embedding for many applications.

For example, employing a reference node embedding is usually quite uncommon for navigation systems because a navigation system usually needs to store the road networks for complete countries or even continents. Furthermore, the network does not only contain the topology of the road network, but also additional information like points of interest or rendering information.

In this paper, we examine possibilities to reduce the memory requirements of a reference node embedding without lowering its performance. Since additional reference nodes increase both the accuracy of the approximation and the memory requirements, the task is strongly coupled with optimizing the performance. Thus, we aim at maximizing the search performance of A^*-search when using an embedding, having a certain predefined memory consumption. To optimize the performance, we identified two possibilities: The first is to select a set of reference nodes that allow close approximations for a majority of the paths in the network. We will formalize the quality of a set of k reference nodes and show that finding an optimal solution is not feasible for larger graphs. Thus, we will propose heuristic methods for finding k reference nodes that allow a high-quality embedding. The second method for decreasing the memory cost is the introduction of sparse embeddings. A sparse embedding reduces the memory costs by storing the embedding information only on a subset of the nodes. As a result the memory requirement of the embedding can be significantly decreased. However, using a sparse embedding does not allow to approximate distances between an arbitrary pair of nodes. Thus, we propose a new shortest path algorithm that is capable to compute shortest paths on a sparse embedded network. Finally, the performance of this algorithm does not only depend on the number of nodes carrying embedding information but is also influenced by the selection of these nodes. Thus, we will propose an algorithm for selecting a certain subset of the nodes that allows the shortest path computation with a rather small overhead compared to A^*-search on a completely embedded network. To conclude, the main contributions of this paper are:

- An examination about choosing reference nodes and a heuristic algorithm for reference node selection.
- An algorithm for shortest path computation that is based on a sparse embedding.
- A method for generating a sparse embedding for a given network that allows efficient shortest path computation.

The rest of the paper is organized as follows. Section 2 surveys related work in the area of embedded spaces and distance approximations based on reference nodes. In section 3, we formalize our setting and the considered problems. Our solutions based on selecting reference nodes and introducing sparse embeddings are described in section 4. Section 5 shows the results of our experimental evaluation comparing memory costs and search performance of the proposed methods. Finally, section 6 summarizes the papers and outline direction for future research.

2. RELATED WORK

Common route search which starts from a single source node to at least one target node is also known as the single-source problem. This problem has been studied very extensively for a long time [6, 2, 4, 5, 9, 10, 11, 21, 30, 37, 40]. Nevertheless, most of the proposed solutions imply some restrictions, like non-negative lengths or weights of links.

If the target of the search is also a single node (in contrast to all shortest path algorithms that compute the shortest paths to all other nodes of the graph [34, 36]), the problem is sometimes also formulated as the point-to-point shortest path problem, as it is for example described in [20, 31, 33, 41].

Another approach of solving the task is to preprocess the graph to obtain some additional information that can be used to enhance subsequent route searches e.g. by providing improved approximations and pruning bounds for the distance computations. By using the precomputed knowledge, the search space can be narrowed on the given graph and thus, the speed of the route search can be improved. Studies using preprocessed data for the detection of approximate shortest paths are for example shown in [3, 23, 38]. Other studies aiming at the extraction of exact paths are using for example geometric information [28, 39], hierarchical decomposition [7], the notion of reach [18] or landmark distances [12] to obtain information that is used during the search process. Landmarks and routing between voronoi cells are proposed in [25] for $k$nn search in spatial network databases. In [24] the method is extended to support moving query objects.

Besides the above mentioned possibilities for precomputation, there also exists the method of embedding the graph into a vector space H using for example the Lipschitz embedding [1, 22]. An embedding is a mapping from the original space into a vector space, where each axis corresponds to a subset of nodes (the reference nodes) of the graph. The coordinate values of an embedded object are the distances to the closest reference object. Embeddings have already been used in skyline queries [32] and point to point searches [26, 13, 15].

One problem that arises with the use of embeddings, is the selection of the subset of nodes which represent the referencing set. [29] use in part work of [1] and select a random subset of nodes to guarantee a lower bound of the embedded distance. [35] adapt the method for spatial networks by projecting vertices into a vector space. Afterwards the Minkowski distance is used to determine

the distance between corresponding representations. This embedding nevertheless is not used to compute shortest paths but an approximate network distance. This distance could be used in subsequent steps to give a better forward approximation in shortest path searches. In [17] an approach is presented which is related to the previous approaches. However it computes exact network distances and shortest paths but implies that edge weights have to be integers.

Another issue is the space that is consumed by the precomputed embedding, if the embedding is calculated for all vertices of the graph. This leads directly to the problem of the selection of applicable embedding nodes which can be translated to graph coverage problems which have also been subject of research like in [19].

3. REFERENCE NODE EMBEDDING REVISITED

In this section, we will formalize our setting, describe the reference node embedding and analyze its memory consumption. We will begin with defining network graphs for representing a road network:

DEFINITION 1 (NETWORK GRAPH). *A network graph is a directed graph $\mathcal{G}(V, E, W)$ with V denoting a set of vertices, $E \subset V \times V$ denoting a set of edges and $W \subset \mathbb{R}_+$ denoting a set of positive edge weights. Since $\mathcal{G}$ is directed, $e = (v_s, v_d) \neq (v_d, v_s) = \hat{e}$. Furthermore, let $\omega : E \Rightarrow \mathbb{R}_+$ be a mapping, assigning a weight ω to each edge $e \in E$.*

In our application, a graph represents a road network and thus, the nodes correspond to crossings, the edges correspond to road segments and the weights describe the considered attributes of each road segment which we call road attributes in the reminder. The attributes of a segment might represent the length, the maximum speed, the time to pass the segment, the number of pedestrian crossings or the maximum ascent etc. In the following, we will assume that all attributes are positive and a small weight is more beneficial than a larger weight.

To calculate a route in a network graph, the most common approach is to calculate the shortest path between a start node v_s and a destination node v_d. One of the most efficient algorithms for shortest path computation is A^*-search. A*-search extends the well-known Dijkstra algorithm by using optimistic distance approximations between the current node v_i and the destination v_d. An optimistic approximation of the distance to the target allows a global pruning of candidate paths. If the combined costs of a path p and the optimistic approximation of the costs between the end of p to v_d are larger than an already encountered path $\hat{p}$ from v_s to v_d, then no extension of p has to be further examined.

For considering the length of each road segment as cost, a simple solution for calculating such a lower bound approximation is to employ Euclidian distance. Since there is no shorter way between two points than the direct line, the Euclidian distance will always be lower than or equal to the distance which has to be traversed when traveling on the road network to the destination. Thus, for this particular cost function the Euclidian distance can be employed to support A^*-search. Unfortunately, this natural lower bound cannot be easily extended to arbitrary other criteria. Thus, for applying A^*-search on an arbitrary cost function, it is necessary to employ a more general approach.

A reference node embedding is a special form of Lipschitz embedding of the traffic network using singleton reference sets which we call *reference nodes*. According to [27] (in [16, 14], these reference nodes are called *landmarks*). The embedding transforms the nodes of a given network graph into k-dimensional vectors, where

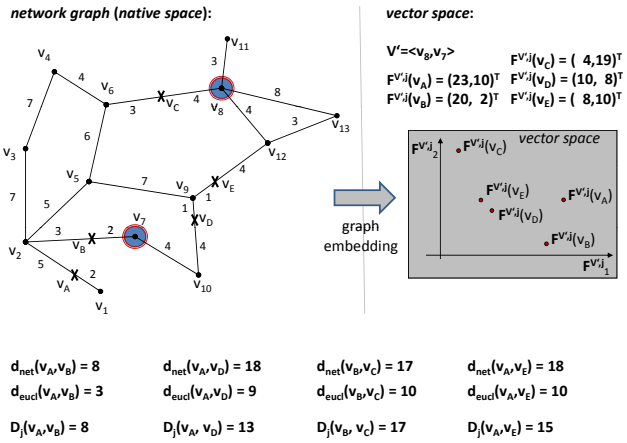


Figure 1: Network graph embedding.

k denotes the number of reference nodes. Therefore, $d_{net}(v_s, v_d)$ denotes $cost(sp_{v_s,v_d})$ where sp_{v_s,v_d} describes the shortest path between the nodes v_s and v_d.

Let $\mathcal{G} = (V, E, W)$ be a network graph and $V' = \langle v_{r_1}, \ldots, v_{r_k} \rangle \subseteq V$ be a subsequence of $k \geq 1$ reference nodes. The embedding, or transformation, of the native space V into a k-dimensional vector space $\mathbb{R}^k$ is a mapping $F^{V'} : V \rightarrow \mathbb{R}^k$, where $|V'| = k$ is the dimensionality of the vector space. A *reference node embedding* of $\mathcal{G}$ based on $V' \subset V$ defines the function $F^{V'}$ as follows:

$$\forall v \in V : F^{V'}(v) = (F_1^{V'}(v), \ldots, F_k^{V'}(v))^{\mathbf{T}},$$

where $F_i^{V'}(v) = d_{net}(v, v_{r_i})$ for $1 \leq i \leq k$ and $d_{net}(v, v_{r_i})$ denotes the network distance between node v and node v_{r_i} according to the corresponding road attribute i.

An example demonstrating the embedding of the network graph using the reference nodes $V' = \langle v_8, v_7 \rangle$ is depicted in Figure 1. Note that our example does not show a complete embedding, but displays the embedding of a subset of graph nodes, i.e. the nodes are displayed using a cross instead of a point. The left graph shows the network graph with edge weights corresponding to an arbitrary road attribute. Nodes, $v_7 \in V'$ and $v_8 \in V'$ are selected as reference nodes. On the right side, the (partial) embedding according to V' is depicted.

DEFINITION 2 (NETWORK DISTANCE ESTIMATION). *Let $\mathcal{G} = (V, E, W)$ and $F^{V'}$ be the reference node embedding of $\mathcal{G}$ w.r.t. $V' \subset V$. For any path $p = (v_s, \ldots, v_d)$, the network distance can be estimated by*

$$D(v_s, v_d) = \max_{i=1..k} |F_i^{V'}(v_s) - F_i^{V'}(v_d)|.$$

In [27] it is shown that the distance $D(v_s, v_d)$ lower bounds the network distance. Thus, we can employ this embedding as lower bound approximation in an A^*-search for arbitrary road attributes.

To generate the embedding for a network graph, we have to compute the network distance for each node to all reference nodes in reverse direction. Since the graph structure is considered to remain fixed, the embedding of the graph nodes can be performed off-line in a preprocessing step. Afterwards, the pre-computed results have to be stored for each of the k reference nodes at each node $v \in V$. Thus, the memory cost of the embedding $Space_{emb}(\mathcal{G})$ is:

$$Space_{emb}(\mathcal{G}) = |V| \cdot k \cdot sizeOf(\text{dist\_type})$$

where $sizeOf$(dist_type) denotes the memory cost required to store a distance. For example, the memory requirement $Space_{emb}(\mathcal{G})$ for a graph having 100,000 nodes, 50 reference nodes and 8 Byte doubles is more than 38 MB.

Let us note that the memory consumption described above is only for a single road attribute. However, in a modern route search algorithm multiple road attributes must be considered. For example if users want to distinguish between the shortest, the fastest, the most ecological or the safest path. In such a case, an embedding for each attribute would be computed and stored. As all embeddings use the same amount of space, the memory consumption is scaling linearly in the number of attributes. In our example, taking the above four road attributes into account, the memory consumption would increase to 152 MB.

Since routing systems usually require various additional information, like points of interest, rendering information etc., memory is often a limited resource and is not easily spent for performance tuning. Especially in the area of small or mobile devices like handheld GPS devices or cell phones, main memory is still a bottleneck and in network applications streaming this type of data might require much higher bandwidth which is also expensive.

4. MEMORY EFFICIENT REFERENCE NODE EMBEDDINGS

In this section, we will introduce our methods aiming at reducing the cost produced by the reference node embedding in terms of memory space and the size of the search space when performing shortest path computations. After shortly discussing the problem on a general level, we will describe our approach leading to our solution for the reference node selection and the introduction of a sparse embedding.

4.1 Reference Node Embedding and Memory Costs

As described in the previous section, the memory costs of the embedding linearly increase with the size of the graph (i.e. the number of nodes $|V|$), the number of reference nodes and the number of attributes[1] used for the edge weights. To a limited degree, we can lower the memory consumption by switching the accuracy of a number from an 8 Byte double value to a 4 Byte float value. However, depending on the given task, this might not be acceptable for many applications due to the loss of accuracy. To conclude, only two feasible approaches remain: Decreasing the number of required reference nodes or decreasing the number of embedded nodes. Solutions according to these two options will be discussed in the next two sections.

4.2 Selecting Reference Nodes

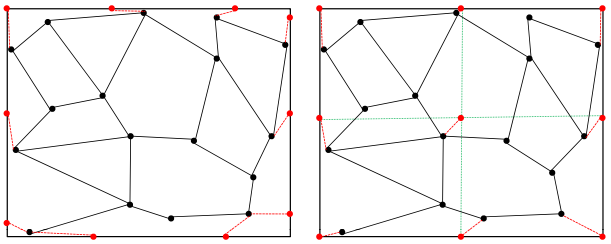
The required number of reference nodes is closely connected to the question which is the best set of reference nodes in the given network. An optimal embedding of size k can be defined by a set of k reference nodes allowing the exact prediction of the remaining distance to the destination for a maximum number of paths. In other words, the number of shortest paths in the graph between nodes v_s and v_d that are a sub-path of a shortest path from either v_s or v_d to at least one reference node R should be maximized. By considering each reference node as the set of the shortest paths it can optimally approximate, we have an instance of the maximum coverage problem [19] which is proved to be NP-hard. In particular, the maximum coverage problem tries to find a k-set from a set of sets where the number of elements in the union of the elements of the k-set is maximized.

Besides the combinatorial problem of finding the optimal k-set, even finding the paths for which the reference node R offers an optimal solution is computationally very expensive. The fastest method to calculate this information for all nodes in the graph has at least the complexity $O(n^3)$ when applying Floyd Warshall's all-pairs-shortest path algorithm [8]. Thus, even the process of determining an optimal embedding is usually very expensive for larger graphs. Note that this is not only an issue of how we defined the optimal embedding. For other optimal embedding definitions, the complexity in combining the gain of various reference nodes remains.

As a consequence of this complexity, computing an optimal set of reference nodes is infeasible even for a medium sized graph. Thus, heuristic approaches have to be applied. A default solution to the problem is to draw a k-sample of all nodes in the graph. This very simple heuristic displayed satisfying result in various previous works [1, 22, 32, 26, 13, 15]. However, with increasing k the probability that the sample contains two reference nodes which add more or less the same information to the embedding (because they are located very close to each other) is constantly increasing. Thus, adding an additional random sample might not significantly add better approximations for a large set of paths.

In the following, we will propose two further reference node selection heuristics leading to a more appropriate embedding. These heuristics are based on the observation that good reference nodes should be as far as possible from all other nodes in the network.

[1]The number of attributes only relates to multi-attribute graphs (MAG) where the weight of an edge (which is related to multiple attributes) is represented by a vector of weights.



(a) Margin heuristic using 8 reference nodes (1 shared node)

(b) Grid heuristic using 9 reference nodes

Figure 2: Heuristics for choosing reference nodes. Red nodes denote auxiliary points used to find the reference nodes which are the nearest neighbors to the auxiliary points. The cardinality of the reference nodes is smaller than the cardinality if different auxiliary nodes share a node of the graph as their nearest neighbor.

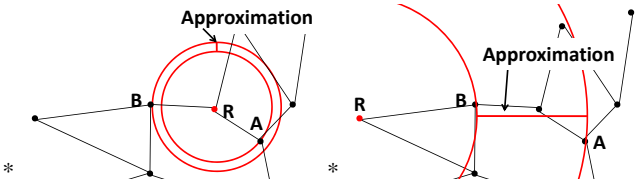


Figure 3: Impact of the spatial position of a reference node R on the approximated minimum distance between the nodes A,B. It can be seen that the approximation performs better, if B is on the shortest path from R to A.

4.2.1 Margin Heuristic

Consider a node v in the graph $\mathcal{G}$ and the tree comprising the shortest path from v to every other node in $\mathcal{G}$. Now let us assume that this expansion tree has a high branching degree but the average length of the path from the root to a leave is rather small. Then the node v should be a central node having a small distance w.r.t. hops to all other nodes. However, if the average number of hops for a shortest path is rather small then the number of sub-paths is limited as well and v does not yield a good approximation for several paths. Thus, a more promising selection of reference nodes are those nodes v having an expansion tree where the average length of the paths from the root to the leaves is rather high. Since having long shortest paths implies the coverage of a large amount of sub-paths, choosing v at the border (margin) of a spatial network yields a good heuristic (cf Figure 3).

To efficiently determine whether a node is placed at the margin of the graph, we can rely on the property of a street network being basically a planar graph with nodes having a spatial position on a two-dimensional surface. Thus, our first heuristic *Margin* proceeds as follows: First determine a minimum bounding rectangle (MBR) around the complete graph. Then, determine the position of k uniformly distributed auxiliary points on the MBR. Afterwards, select the closest node in $\mathcal{G}$ for each auxiliary point and mark it as a reference node for the embedding. This results in at most k reference node. If different auxiliary points share a common node as their closest node, the amount of reference nodes will be smaller than k. See Figure 2(a) for a visualization of the *Margin*-heuristic and the case of a shared node.

As can be seen in our experimental evaluation, this heuristic yields a graph embedding with a significantly better approximation accuracy than the mentioned random heuristic. Especially, for small values of k, the *Margin*-heuristic yields excellent results. However, for large values of k, we observed that *Margin* practically could not increase the approximation accuracy by selecting additional reference nodes. In comparison, the simple random heuristic still achieved improvements with increasing k. The reason for this effect can be explained easily: If the number of reference nodes approximates the number of available reference node candidates (i.e. number of margin nodes), then additional selection of further reference nodes does not provide new relevant information for the network distance approximations. However, there might exist reference node candidates in the inner areas of the graph that would be more helpful for the embedding. For this reason, the random heuristic tends to outperform the *Margin*-heuristic when further increasing the number of reference nodes after the number of reference nodes exceeds the number of margin nodes.

4.2.2 Grid Heuristic

To overcome the above mentioned problem of the *Margin*-heuristic, we propose a second heuristic called *Grid*-heuristic. This heuristic proceeds similar to the *Margin*-heuristic, i.e. it defines auxiliary points and selects nodes as reference nodes that are closest to these auxiliary points. The difference to the *Margin*-heuristic is that it does not only consider auxiliary points at the border of the graph and MBR but also inside the MBR on all vertices of the grid (cf Figure 2(b)). Due to the uniform nature of the grid, points are also distributed uniformly on the border of the grid. However, for considerable large values of k the heuristic adds more auxiliary points within the rectangle allowing the discovery of new useful reference nodes.

Comparing *Margin*- and *Grid*-heuristic with the same value of k, it is obvious that the saturation of reference nodes on the border of the graph is much lower for the *Grid*-heuristic which also means that the *Grid*-heuristic stores less redundant information than the *Margin*-heuristic.

4.3 Shortest Path Computation based on a Sparse Embedding

The second important aspect to be considered when trying to reduce the memory cost of a graph embedding is the number of nodes that are embedded, i.e. the number of nodes storing the embedding information. Common graph embedding approaches apply the embedding to each node of the graph, i.e. they compute and store for each node of the graph the network distances to all reference nodes which leads to high memory cost. The question at issue is, whether we really need an embedding for each node of the graph in order to estimate the network distance or to compute the shortest path between two nodes of the graph in an efficient way. This question arises due to the following observation: Given two adjacent nodes v_i and v_j and the weight w of the edge connecting these two nodes, then the graph embedding information assigned to these two nodes are quite redundant, as the corresponding network distances to the reference nodes do not differ more than w. This leads to the assumption that the embedding cost might be reduced by embedding only one of the nodes (e.g v_i) and approximate the embedding of v_j online by taking into account the embedding of v_i and the weight of the connecting edge w. Based on this observation, in the following, we will introduce a shortest path algorithm[2] based on a graph embedding which is applied only to a sample of graph nodes.

The basic idea of our method is that it is not necessary to update the optimistic approximation of the distance to the target node with each additional node. Instead, it is allowed to keep the maximum of the distance traversed so far and the approximation of the predecessor path. This way each new path can still be ranked w.r.t. an optimistic approximation to the target and thus, A^*-search will process the shortest path still correctly. Of course, the approximation of a path that computes its approximated length by this simple trick will be too small for the majority of paths. Thus, it is possible that the algorithm visits additional nodes due to a bad approximation. However, bad approximation might happen for most approximation heuristics anyway. Furthermore, a bad approximation does not necessarily cause any additional cost, if the path ranked too high would be extended anyway. Especially, in cases were there is only one or two successor paths working with a too optimistic approximation is not a big problem. As long as further extensions of a path allow calculating a more accurate forward approximation often enough, the overhead of paths is kept considerably small.

A sparsely embedded network can be formalized in same way as a completely embedded network with the difference that it is not possible to access $F^{V'}(v)$ for all nodes $v \in V$. Instead, there exists only a subset $S \subset V$ for which $F^{V'}(s)$ is available. Distance approximation by a sparse embedding can be formalized as follows:

DEFINITION 3 (DISTANCE APPROXIMATION). *Let $\mathcal{G} = (V, E, W)$ and $F^{V'}$ be the reference node embedding of $S \subset V$ w.r.t. the reference nodes $V' \subset V$. For each path $p = (v_s, \ldots, v_t, v_d)$, the network distance can be estimated by*

$$D(v_s, v_d) = \begin{cases} \max_{i=1..k} |F_i^{V'}(v_s) - F_i^{V'}(v_d)| & if \quad v_s, v_d \in S \\ 0 & else \end{cases}.$$

A consequence of the sparse embedding is that it is not possible to generate approximations for any pair of nodes in the graph.

[2]Network distance estimations between two arbitrary graph nodes, based on the sparse embedding are included.

For the problem of the missing embedding of the current location, the solution is to reuse the approximated distance of the last node on the path having an embedding. Formally, given a path $p = (v_1, ..., v_k)$ and its optimistic approximation $approx(p) = d_{net}(v_s, v_k) + D(v_s, v_k)$ for the shortest extension $ext(p, v_d) = (v_1, .., v_k, .., v_d)$ to the target node n_d. Then $approx(p) \leq approx(\hat{p})$ holds for any extension $\hat{p} = (v_1, .., v_k, v_{new})$. The reason for this observation is quite obvious: If v_{new} is on the shortest path between v_k and v_d then $approx(p)$ must still be smaller than the cost of the shortest path between v_k and v_d because it is an optimistic approximation. If v_{new} is not on the shortest path between v_k and v_d, then the path q between v_k and v_d crossing v_{new} ($q = (v_k, v_{new}, .., v_d)$) must have larger costs than the shortest path between v_k and v_d ($sp(v_k, v_d)$). Since $approx(p) \leq sp(v_k, v_d)$, it follows that $approx(p) < cost(q)$ as well. Thus, $approx(p)$ lower bounds the costs of all extensions of p as well.

Due to this property, it is allowed to run A^* using this approximation. However, since the lower bound approximation must not grow with each additional hop, the search space of the algorithm will increase to a certain extend. Let us note that we can additionally consider the current cost of q in comparison to $approx(p)$ and use the larger value as approximation. This optimization is sometimes beneficial if v_k is already quite close to v_d. The second problem is that it is not possible to make any approximation of the remaining distance to v_d if v_d is not embedded. To solve this problem, we proceed as follows. Before starting the traversal from v_s to v_d, we explore the neighborhood of v_d and collect the so-called surrounding $surround(v_d)$. The idea of the surrounding is to find a set of embedded nodes $border(v_d) \in surround(v_d)$ for which it is guaranteed that the shortest path from each node $v_{out} \notin surround(v_d)$ to v_d has to contain at least one member $v_b \in border(v_d)$. Thus, v_s is either part of $surround(v_d)$ or can be composed by concatenating the paths $sp(v_s, v_b)$ and $sp(v_b, v_d)$. To compute $surround(n_d)$ and $border(v_d)$, we perform a breadth-first traversal in reverse direction beginning with v_d. If a reverse path starting with an embedded node v_{emb} and leading to n_d is found, we add v_{emb} to $border(v_d)$ and store the cost of the path $sp(v_{emb}, v_d)$. $border(v_d)$ is complete when there is no path in the search left that ends with a node that is not embedded. However, at this point of time it is possible that we did not find the shortest path to all elements of $border(v_d)$. Thus, we have to continue and extend the search in all directions up to a distance of $\max_{v_b \in border(v_d)} sp(v_b, v_d)$ to guarantee to find out the real distance $d_{net}(v_b, v_d)$ for any $v_b \in border(v_d)$. To cover the case that v_s is already encountered during this traversal, we can treat v_s in the same way as the elements of $border(v_d)$. Let us note that in this case the search is basically a Dijkstra search guaranteeing the correct result but not yielding a high efficiency. However, having a proper selection of embedded nodes S, this case can be prevented for any pair of nodes having a reasonable distance between each other. The surrounding of $surround(v_d)$ is given by the set of nodes which are visited during the first traversal for finding all elements of $border(v_d)$.

If $v_s \in surround(v_d)$ the algorithm has already found the shortest path between v_s and v_d and the search terminates. In the more common case that $v_s \notin surround(v_d)$, we now an A^*-like best first search beginning with v_s to all elements of $border(v_d)$. However, to rank a candidate path $p = (v_s, \ldots, v_k)$ in the queue of the best first search, we employ the following function:

$$approx_{surround}(p) = \min_{v_b \in border(v_d)} \{sp(v_s, v_k) + D(v_k, v_b) + sp(v_b, v_d)$$

$approx_{surround}(p)$ approximates the shortest distance between v_s and v_d via v_k by splitting it into three segments for each node v_b. The first is the true cost between $d_{net}(v_s, v_k)$ which has been already computed during the traversal. The second is the optimistic approximation between v_k and v_b. The last segment is the true cost of $sp(v_b, v_d)$ which has been calculated during the search for $border(v_d)$. Since we do not know which of the $v_b \in border(v_d)$ is indeed on the shortest path between v_s and v_d, we have to take the minimum in order to lower bound the true distance. In the case that $v_k \in border(v_d)$, we found a complete path between v_s and v_d and $approx_{surround}(v_d)$ displays the actual costs of that path. Thus, in the case that such path occupies the top of the queue, we can guarantee that it is a shortest path.

To conclude, our algorithm proceeds as follows:

- Determine $surround(n_d)$
- Begin a best first traversal of $\mathcal{G}$ with n_s as start. Each path $p = (n_s, .., n_k)$ is weighted by the maximum of $cost(p)$ and $approx_{surround}(n_k, n_d)$.
- If a path $p = (n_s, .., n_{sur})$ with $n_{sur} \in surround(n_d)$ is on top of the priority queue, return $p, sp(n_{sur}, n_d)$ as the shortest path.

4.4 Generating a Sparse Embedding

The shortest path algorithm described above computes correct shortest paths for any sparsely embedded network. However, in the extreme case the shortest path is not computed using A^*-search, but the plain reverse Dijkstra approach for computing the surrounding of n_d. Thus, in order to achieve comparable search times to $A*$-search on a complete embedding, it is important to select a suitable set of nodes in the graph to store the embedding information. In general, a basic approach is to choose a random sample, i.e. chose every j-th node. Since a random sample obviously does not take the topology of the network into account, we will propose another more directed method for finding a suitable set of embedded nodes. In order to keep the overhead of the proposed algorithm compared to A^*-search on a completely embedded graph as small as possible, we have to analyze in which cases the missing embedding information causes the most overhead. Furthermore, we have to keep in mind that the overhead and the memory consumption are concurrent goals. Thus, the goal of our algorithm is to find a selection of nodes that causes the smallest overhead for a given amount of memory that is available for the embedding.

Theoretically, it is possible to measure the overhead for a certain set of reference nodes by comparing the nodes being accessed for an A^*-search on a completely embedded network in comparison to a sparse embedded network for all possible start and target nodes. However, the effect of a missing embedding information is strongly dependent on the other embedded nodes in the network. If a node does not have an embedding but all its neighbors have, the caused overhead is usually much smaller than in the case that there is no embedded node in the complete neighborhood. As a result, to determine the best possible sparse embedding comprising k nodes would require to test the overhead for all paths for all subsets of nodes V having k elements. Since this approach is too expensive for even smaller graphs, we will propose heuristic algorithms.

There are two reasons for the overhead being caused by a sparse embedding. The first is the size of the surrounding. If there are too few embedded nodes, determining $surround(n_d)$ will visit too many nodes in the graph. In the extreme case, the start node n_s is already visited in the step. In this case, the proposed algorithm degenerates to a reverse Dijkstra search from the target node to the

start node. The second problem is a smaller forward approximation for the paths not ending with an embedded node. Again if the density of embedded nodes in the graph becomes too low, the search algorithm will more and more tend to judge paths mainly based on their cost and the importance of the forward approximation becomes significantly smaller. Thus, the algorithm again converges to a Dijkstra-like search.

To find a good embedding and avoid these problems, we can formulate some requirements of a good embedding:

- Consider the node $n \notin V_{embed}$, then there is no path $p = (n_1, .., n_k)$ with $cost(p) > \epsilon$ and $\nexists n_i \in V_{embed}$ for $1 \leq i \leq k$.
- For each node n $|surround(n)| < g^n$ where each g is the maximum degree of any node in $G(V, E)$ and n is a parameter describing an branching level.

While the first requirement makes sure that there is no path that is extended too often without any new approximation, the requirement directly controls the size of the surrounding. In the following, we will propose an algorithm for determining a subset of nodes for the embedding that guarantees both requirements. However, since it is an heuristic algorithm, we cannot guarantee that the node having embedding information has a minimal cardinality.

The idea of our algorithm is to successively check for each node whether there is a close enough embedded node in each of its directions. Thus, we traverse the graph in each direction and stop if the path does not fulfill the above requirements or contains a node being already selected for the embedding. If the search terminates in each direction because of a node being already selected for the embedding, the node remains without an embedding. If there is at least one path not being extended because of the above requirements, we select the path with largest sum of node degrees and add its start and its end node to the list of embedded nodes.

To process all nodes in a determined order, we process nodes having a larger degree before nodes having a smaller degree. The idea behind considering the degree is that both sources of a search overhead strongly react to a large number of successors. A large surrounding causes larger costs for its traversal and additionally requires more time during the A^*-like traversal. Additionally, if we cannot not determine a new better approximation for a path, an end node having less outgoing edges will generate less unnecessary extensions and thus, less overhead.

To conclude, our algorithm works as follows:

- Calculate the full embedding for all $v \in \mathcal{G}$
- initialize an empty set for covered nodes: $covered$
- Sort all nodes $v \in \mathcal{G}$ in a queue Q by their degree in decreasing order as nodes with a high degree are more important to be embedded than nodes with a lower degree.
- $\forall v \in Q \wedge v \notin covered$:
  - identify the shortest path $p = (v_1, ..., v_k)$ starting at v to any uncovered node with respect to the following constraints:
    $length(p) > \epsilon \wedge$
    $\forall v \in p : v \notin covered \wedge$
    $\forall v \in v_{2,...,k-1} : degree(v) \leq 2$
  - Add $v_1, v_k \in p$ to $covered$.
- remove the embedding information from all $v \notin covered$

5. EXPERIMENTAL EVALUATION

In this section, we present our experimental evaluation describing the trade-off between memory and search performance. Our experiments were performed on two road networks. The first network comprising 6105 nodes represents the German city of Oldenburg. The second network having more than 171,000 nodes displays consists of major roads in Northern America. All experiments where performed on a 32 bit workstation having 2 GB main memory and an AMD Athlon 5000+ processor. All algorithms were implemented in Java 1.6.

All experiments where performed on a sample of 400 test queries randomly selected by 20 starting and 20 destination nodes being drawn by an uniform distribution. To measure the cost of a shortest path computation, we counted the number of accessed nodes during the traversal. The memory costs were calculated by assuming that each distance value in the embedding is represented by an 8 Byte double value. All experiments were performed multiple times to rule out outlier effects in the involved random processes.

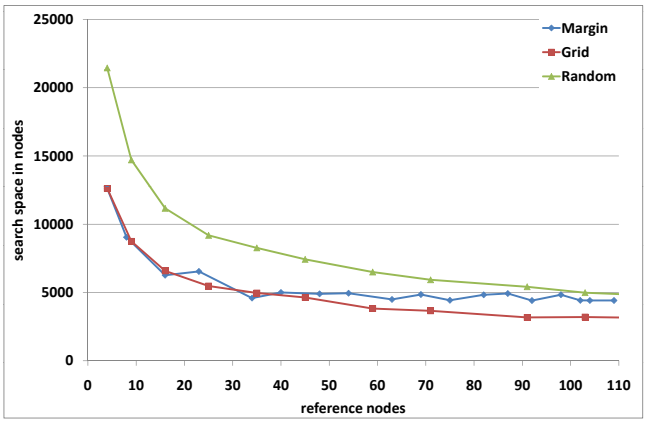
5.1 Influence of Reference Nodes

In a first set of experiments, we compared the performance for three different heuristics for selecting a set of k reference nodes for varying numbers of k based on an completely embedded graph. The first heuristic is called $Random$ and draws a uniform sample from the nodes V. The second method is *Margin*, selecting only reference nodes being close to the margin of the graph. The third heuristic is the *Grid* heuristic, selecting reference nodes being close to auxiliary points being placed on a uniform grid.

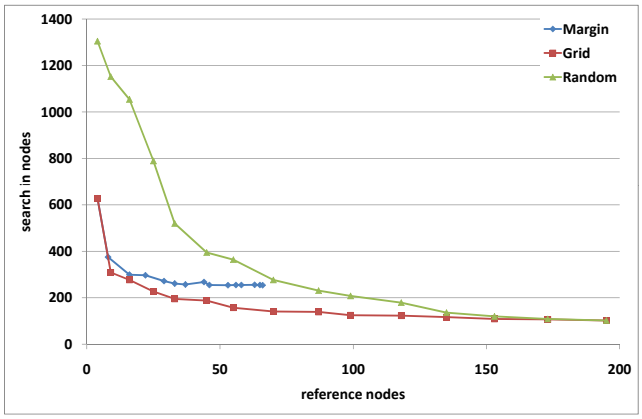
Figure 5 displays the results on both datasets. While the x-axis displays the memory costs of a complete embedding w.r.t. k reference nodes. The y-axis displays the performance measured by the amount of the graph being visited during the A^*-search. Let us note that we performed this set of experiments using artificially generated edge weights in order to rule out the possibility that our heuristics are only valid for using the length of each edge for weighting. For both datasets, it can be observed that for small values of k both $Grid$ and $Margin$ considerably outperform the random selection of reference nodes. The reason for the bad performance of $Random$ is that this heuristic does not try to select nodes being useful reference nodes yielding good approximations for many paths. Furthermore, there is no mechanism to reduce the selection of reference nodes yielding good approximation for the same set of paths. Only after strongly increasing the number of reference points the general approximation quality increases to a reasonable level allowing an fast computation of shortest paths with A^*-search. The performance of $Grid$ and $Margin$ is rather comparable for small values of k. Furthermore, since both heuristics try to distribute reference nodes around the margin of the graph the performance strongly increases very quickly for small values of k. However, after the performance reaches a certain level the $Grid$ heuristic is not capable to achieve any additional performance advantage. The reason for this effect is that the margin of the network is covered quite well by reference nodes. Therefore, adding additional nodes at the margin does not allow to make better approximations for a large amount of paths. In contrast the $Grid$ heuristic selecting additional nodes from the inner regions of the graph still yields performance advantages even if the number of reference nodes k increases to rather large values. Thus, selecting reference nodes in the inner regions of the graph yields significant performance advantages.

5.2 Influence of Embedded Nodes

In the second set of experiments, we tested the performance decrease when reducing the number of embedded nodes in the net-

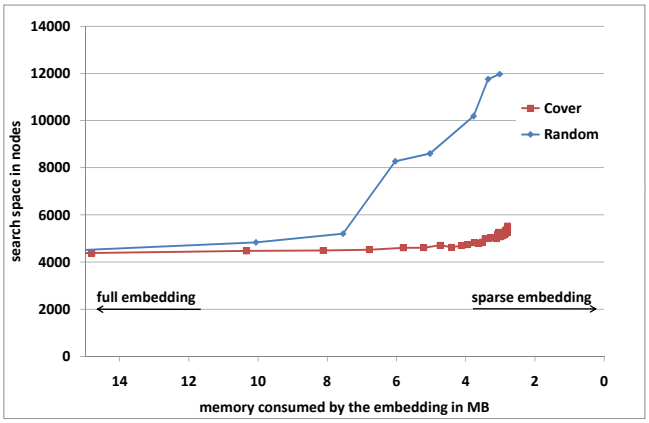


(a) Evaluation based on the North America graph.

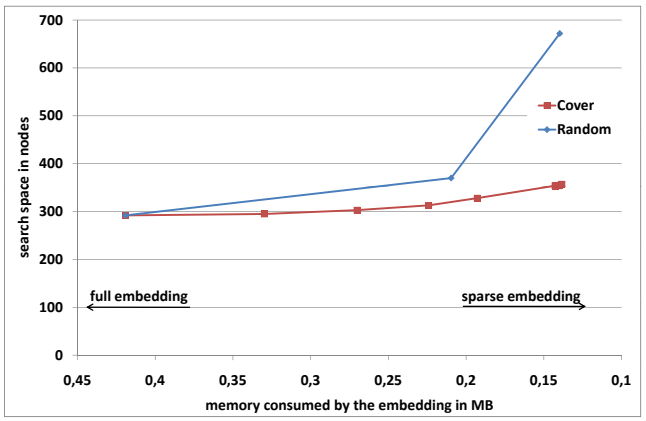


(b) Evaluation based on the Oldenburg graph.

Figure 4: Evaluating the influence of reference nodes using different heuristics shows that the grid heuristic converges faster and more stable than $Random$ or $Margin$. $Margin$ cannot decrease below a certain level as soon as (almost) all margin nodes of the graph have been selected as reference nodes.



(a) North America graph using 45 reference nodes.



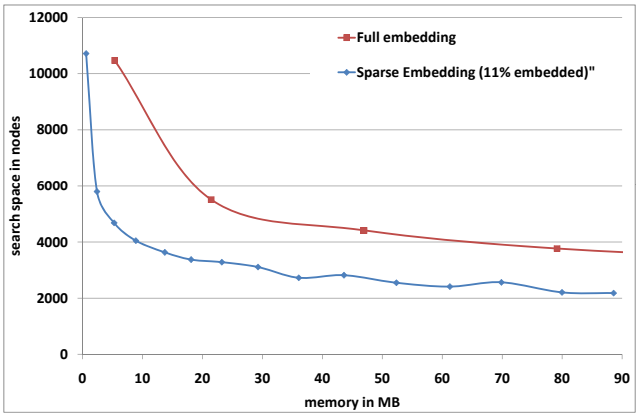
(b) Oldenburg graph using 9 reference nodes.

Figure 5: The diagrams show the amount of search space w.r.t the amount of embedding information. The rightmost values denote the search space with a full embedded of the graph. Removing the embedding using the proposed algorithm causes a considerably lower increase of the search space than removing the embedding randomly.

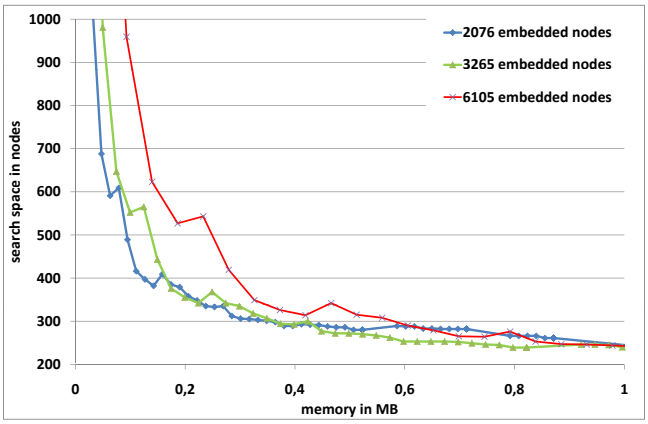
works. Therefore, we selected 9 reference nodes for the Oldenburg graph and 45 reference nodes for Northern America network employing the grid heuristic. We compare our proposed method with randomly selecting every ith node in the network for carrying embedding information. The results are displayed in Figure 5. In both data sets the performance decrease compared to an complete embedding is still tolerable when embedding about half of the network with both of the heuristics. However, when embedding less than half of the network, randomly selecting nodes quickly loosed a lot of performance compared to an complete embedding. In contrast, our solution still keeps the overhead at a value of about 10% when embedding 10% the data in the North America network. For the Oldenburg network, we could observe a 22% overhead for the search space when embedding 33% of the network. In comparison, the random heuristic displays an overhead of 130% for the same percentage of embedded nodes. To conclude, using the proposed selection algorithm, the performance loss when embedding only one half of the nodes was ca. 3% for the North America network and 7% for the smaller but more dense Oldenburg network.

5.3 Memory Trade-Off

In a final set of experiments, we combine both approaches in order to examine the trade-off between the number of embedded nodes and reference nodes. Thus, we measure the performance when constantly increasing the number of reference nodes selected by the $Grid$ heuristic for a completely embedded and a sparsely embedded graph where the selection has been done by our new algorithm. To compare the combinations, we determine the amount of memory required to store the complete embedding. Figure 6 displays the results on both networks. On the x-axis we display the memory requirements in MB while in the y-axis the number of accessed nodes is displayed. For the Oldenburg graph, we compared the complete embedding to a sparse embedding of 33% and 50% of the nodes. For the North America network, we even employed an embedding only comprising about 11% of the nodes. For a rather small amount of memory it can be seen that the solutions using a sparse embedding significantly outperform the complete embedding on less reference nodes. The reason for this effect is that the amount of memory is too small for storing a complete embedding



(a) North America graph with 11% embedded nodes and 4 - 587 reference nodes.



(b) Oldenburg graph with 100%, 50% and 33% embedded nodes and 2 - 87 reference nodes.

Figure 6: The diagrams show the impact of the number of reference nodes on the search space. The x-axis shows the amount of memory that is used by the embedding for 4 - 587 reference nodes on the North America graph and 2 - 87 reference nodes on the Oldenburg graph.)

w.r.t. a sufficiently large number of reference nodes. Thus, adding an additional reference node can still significantly increase the performance. However, to allow this additional nodes, the embedding must be made more sparse to free the required memory. With an increasing amount of available memory, the number of reference nodes that can be employed in the complete embedding reaches a level that allows excellent approximation. Thus, freeing memory by using a sparse embedding for additional reference nodes does not compensate for the overhead of using a sparse embedding. However, since memory is a rather limited resource in applications handling large networks, it is necessary to employ both methods to maximize the performance for the amount of memory that can be spared for the reference node embedding.

6. CONCLUSION

In this paper, we examined distance approximations in arbitrary attributed graphs using the reference node embedding proposed by [1, 22]. A reference node embedding allows the efficient calculation of optimistic approximations for the distance between to nodes w.r.t. to arbitrary positive edge weights. Thus, the embedding is applicable to any type of attribute in various types of applications. A large drawback of this approach is the memory consumption depending on the number of reference points that are necessary to achieve a sufficiently good approximation quality. To address this problem, we first of all try to reduce the number of reference nodes which are necessary to achieve a sufficiently good approximation. Therefore, we propose two heuristics called $Margin$ and $Grid$. Furthermore, we introduce sparse embeddings which do not store embedding information for each node in the network. However, since it is not possible to make distance approximations for all pairs of nodes in a sparse embedding, we show that using the sparse embedding can be used for efficient shortest path computation. Thus, we propose a new algorithm which is a hybrid between Dijkstra's algorithm and A^*-search depending on the portion of the graph which carries an embedding. Since the selection of the nodes carrying the embedding information plays an important role for the performance of the algorithm, we finally propose an algorithm for selecting the subset of the node that should carry embedding information. In our experimental, evaluation we show the behavior of our method w.r.t. performance and memory consumption. Furthermore, we examine the combination of reference nodes and embedded nodes for various amounts of available main memory.

For future work, we will examine the use of a sparse embedding for other problems than shortest path computation and effects on systems storing extremely large network on secondary storages.

Acknowledgments

This research has been supported in part by the THESEUS program in the MEDICO and CTC projects. They are funded by the German Federal Ministry of Economics and Technology under the grant number 01MQ07020. The responsibility for this publication lies with the authors.

7. REFERENCES

[1] J. Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1):46–52, 1985.

[2] B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. In *In Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, 1994.

[3] L. J. Cowen and C. G. Wagner. Compact roundtrip routing in directed networks. In *In Proc. Symp. on Principles of Distributed Computation*, 2000.

[4] G. B. Dantzig. Linear programming and extensions. In *Princeton Univ. Press, Princeton, NJ*, 1962.

[5] E. V. Denardo and B. L. Fox. Shortest-route methods: 1. reaching, pruning, and buckets. In *Oper. Res.*, 1979.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Numer. Math.*, 1959.

[7] D. W. F. Schulz and K. Weihe. Using multi-level graphs for timetable information. In *In Proc. 4th International Workshop on Algorithm Engineering and Experiments, volume 2409 of Lecture Notes in Computer Science*, 2002.

[8] R. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.

[9] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *J. Assoc. Comput. Mach.*, 1987.

[10] G. Gallo and S. Pallottino. Shortest paths algorithms. In *Annals of Oper. Res.*, 1988.

[11] A. V. Goldberg. A simple shortest path algorithm with linear average time. In *In Proc. 9th Annual European Symposium on Algorithms*, 2001.

[12] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *Technical Report MSR-TR-2004-24, Microsoft Research*, 2004.

[13] A. V. Goldberg, H. Kaplan, and R. F. Werneck. Reach for a*: Efficient point-to-point shortest path algorithms. In *ALENEX06*, 2006.

[14] A. V. Goldberg, H. Kaplan, and R. F. Werneck. "Reach for A*: Efficient point-to-point shortest path algorithms". In *Proc. of the 8th WS on Algorithm Engineering and Experiments (ALENEX), SIAM*, 2006.

[15] A. V. Goldberg and R. F. Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX05*, 2005.

[16] A. V. Goldberg and R. F. Werneck. "Computing Point-to-Point Shortest Paths from External Memory". In *Proc. of the 7th WS on Algorithm Engineering and Experiments (ALENEX), SIAM*, 2005.

[17] S. Gupta, S. Kopparty, and C. Ravishankar. Roads, codes, and spatiotemporal queries. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2004.

[18] R. Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *In Proc. 6th International Workshop on Algorithm Engineering and Experiments*, 2004.

[19] D. S. Hochbaum. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. 1997.

[20] T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh. A fast algorithm for finding better routes by ai search techniques. In *In Proc. Vehicle Navigation and Information Systems Conference. IEEE*, 1994.

[21] R. Jacob, M. V. Marathe, and K. Nagel. A computational study of routing algorithms for realistic transportation networks. In *Networks. Oper. Res.*, 1962.

[22] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemp. Math.*, 26:189–206, 1984.

[23] P. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *In Proc. 13th ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[24] M. Kolahdouzan and C. Shahabi. Continuous k-nearest neighbor queries in spatial network databases. In *Proc. of STDBM*, 2004.

[25] M. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, 2004.

[26] H.-P. Kriegel, P. Kröger, P. Kunath, M. Renz, and T. Schmidt. Proximity queries in large traffic networks. In *Proceedings of the 15th ACM International Symposium on Advances in Geographic Information Systems (ACM GIS), Seattle, WA*, 2007.

[27] H.-P. Kriegel, P. Kröger, P. Kunath, M. Renz, and T. Schmidt. "Proximity Queries in Large Traffic Networks". In *Proc. 15th Int. Symposium on Advances in Geographic Information Systeme (ACM GIS'07), Seattle, WA*, 2007.

[28] U. Lauther. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In *In IfGIprints 22, Institut fuer Geoinformatik, Universitaet Muenster*, 2004.

[29] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[30] U. Meyer. Single-source shortest paths on arbitrary directed graphs in linear average time. In *In Proc. 12th ACM-SIAM Symposium on Discrete Algorithms*, 2001.

[31] I. Pohl. Bi-directional search. In *In Machine Intelligence, Edinburgh Univ. Press*, 1971.

[32] M. Schubert, M. Renz, and H. Kriegel. Route skyline queries: A multi-preference path planning approach. In *Proceedings of the 26th International Conference on Data Engineering (ICDE), Long Beach, CA*, 2010.

[33] R. Sedgewick and J. S. Vitter. Shortest paths in euclidean graphs. In *Algorithmica*, 1986.

[34] R. Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, 1992.

[35] C. Shahabi, M. Kolahdouzan, and M. Sharifzadeh. A road network embedding technique for k-nearest neighbor search in moving object databases. *GeoInformatica*, 7(3), 2003.

[36] P. Spira. A new algorithm for finding all shortest paths in a graph of positive arcs in average time $o(n^2 log^2 n)$. *SIAM Journal on Computing*, 2:28, 1973.

[37] M. Thorup. Undirected single-source shortest paths with positive integerweights in linear time. In *J. Assoc. Comput. Mach*, 1999.

[38] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In *In Proc. 42nd IEEE Annual Symposium on Foundations of Computer Science*, 2001.

[39] D. Wagner and T. Willhalm. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *In Proc. 11th Annual European Symposium on Algorithms, volume 2832 of Lecture Notes in Computer Science*, 2003.

[40] F. B. Zhan and C. E. Noon. Shortest path algorithms: An evaluation using real road networks. In *Transp. Sci.*, 1998.

[41] F. B. Zhan and C. E. Noon. A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths. In *Journal of Geographic Information and Decision Analysis*, volume 4, 2000.