Least Squares Approximate Distance Oracles for Spatial Networks

Joris Maervoet[1], Jan Christiaens[1], Patrick De Causmaecker[2,3], and Greet Vanden Berghe[1,3]

[1] KU Leuven, Department of Computer Science, CODeS,
Gebr. De Smetstraat 1, 9000 Ghent, Belgium
[2] KU Leuven-Kulak, Department of Computer Science, CODeS,
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
[3] iMinds - ITEC - KU Leuven

Abstract. An *approximate distance oracle* is a compact data structure or model that is able to answer network distance queries between any two vertices of a graph. Distance oracles are of particular interest for applications that require real-time network distance approximations and only have limited time to build or space to store a predictive model. A conventional distance oracle employs a guaranteed upper and lower bound to the relative error of the approximation and often pegs down its required size on these bounds. These bounds do however not clearly represent the overall performance of the oracle, and the relaxation of these bounds can produce a reduced complexity of the precalculation time, the oracle space and the query time. The present paper introduces the concept of a distance oracle that minimizes the root-mean-square error of its network distance approximations. An appropriate distance oracle of size $O(1)$ is presented for both general and spatial networks. A more advanced least squares approximate distance oracle is introduced for spatial networks. It is based on clustering graph vertices that share similar shortest paths starting or ending in these vertices. Its prediction accuracy is demonstrated in the context of travel time approximation in a transportation network for vehicle navigation.

Keywords: distance oracle, spatial networks, geographical data processing.

1 Introduction

As introduced by Thorup and Zwick [1], an *approximate distance oracle* (ADO) is a data structure that immediately answers network distance queries between any two vertices of a graph. In contrast to data structures arising from precomputing the distances between any pair of vertices of the graph, ADOs require limited space and can be built in limited computational time. Real-time distance approximation is of particular interest for applications such as calculating the distance between two people in a social network or retrieving the distances from the current location to a set of alternative locations [2], e.g. restaurants, in a

transportation network. In these domains an estimate of the network distance is often sufficient.
Many combinatorial optimization problems from logistics, public transportation and tourism, require a transportation cost matrix for a set of locations. The vehicle routing problem [3], for instance, is about finding the path of lowest cost from one or several depots visiting a set of customer locations. The goal of the orienteering problem [4] is to find a path, of a travel cost below a given threshold and along a selection of locations, maximizing the sum of collected location scores. The matrix mentioned above contains the transportation cost, often travel time or fuel cost, between any origin-destination pair of the problem in a transportation network. Using an ADO to determine this matrix is advantageous when the problem's locations often change and the transportation cost calculation must be offered as a portable software component.

We now introduce the following definitions. A graph, further also referred to as a general network, is denoted $G = (V, E)$, where V is the set of vertices and E the set of edges of the graph. The number of vertices $|V| = n$ and the number of edges $|E| = m$. By default, G is a *weighted* graph, which means that any edge $e \in E$ has a weight $w(e) > 0$. A path between the vertices u and v in G is a sequence of edges consecutive in G, starting in u and ending in v. Its cost equals the sum of the edge weights. The *network distance* $d_G(u, v)$ is the cost of the path of lowest cost between u and v in G. A spatial network is a general network where each $v \in V$ corresponds to a position $p(v) \in \mathbb{R}^d$. The *spatial distance* $d_S(u, v)$ is a function of $p(u)$ and $p(v)$ in $\mathbb{R}^d$. An Euclidean network is a spatial network where the weight of any edge e connecting to vertices u and v is defined as $w(e) = d_S(u, v)$, with $d_S(u, v)$ taking the Euclidean distance between $p(u)$ and $p(v)$. Distances in this type of network satisfy the triangle inequality i.e. $\forall u, v, w \in V : d_G(u, w) \leq d_G(u, v) + d_G(v, w)$, and the network distance has a lower bound: $\forall u, v \in V : d_G(u, v) \geq d_S(u, v)$.

In any of these networks, an ADO is a data structure or model that answers network distance queries $d_G(u, v)$ by a *reported distance* $d_R(u, v)$. Conventional ADOs assume a guaranteed upper and lower bound to the error of the reported distance. In this context, a distance oracle has a *stretch factor* α when $\forall u, v \in V : d_G(u, v) \leq d_R(u, v) \leq \alpha \cdot d_G(u, v)$. Alternatively, an ADO is called ϵ-approximate when the relative error of any approximation generated by the oracle is not more than ϵ i.e. $\forall u, v \in V : abs(d_G(u, v) - d_R(u, v))/d_G(u, v) \leq \epsilon$. Thorup and Zwick [1] showed for any integer k, that building an oracle of stretch $2k-1$, answering queries in k time, requires a space of at least $n^{1+1/k}$. Sommer et al. [5] argue that this bound does not provide useful information for sparse graphs, and prove for any ADO that a space of at least $n^{1+\Omega(1/t\alpha)}/lg(n)$ is required to build an oracle of stretch α and query time t. Network distance approximation for spatial networks can be seen as fine-tuning "as the crow flies" distances towards network distances. ADOs for spatial networks exploit the spatial coherence of source and destination vertices of similar shortest paths. Sankaranarayanan and Samet [2] introduce an ϵ-approximate oracle of this type based on well-separated pair decomposition in a d-dimensional space. The oracle's space requirements are

$O(n/\epsilon^d)$ answering queries in time $O(log(n))$. Sankaranarayanan and Samet [2] conducted an experiment on a publicly available US transportation network with distance-weights. It confirmed the linear storage requirements and yielded an average relative error of 0.9% for $\epsilon = 0.1$, which is 10%. With regard to Euclidean networks, Gudmundsson et al. [6] refer to a number of solutions based on polyhedral surfaces and obstacles. They notice that none of these can be used to build an ADO of subquadratic space answering queries in constant time. They present an oracle for Euclidean networks with $m = O(n)$ of size $O(n \cdot log(n))$ and query time $O(1)$. It is based on partitioning the graph into a set of clusters with a fixed radius. Mainly for networks satisfying the triangle inequality, many variants of the *landmark embedding* technique have been proposed. It involves that a set of landmarks R is selected from V and that the network distances between any vertex and V and one or more vertices in R are computed. Real-time network distance approximation is based on these precomputed distances. In the ADO proposed by Qiao et al. [7], each landmark covers the set of vertices located within a given radius. This radius can be linked to the upper bound of the absolute approximation error. They show that finding the minimal set of landmarks is NP-hard. Qiao et al. [8] improve the approximation accuracy significantly by looking up the *least common ancestor* (LCA) of the two queried vertices i.e. the last vertex shared by the paths of lowest cost starting in a global landmark, in efficient indices.

In what follows, we raise several concerns in regard to the practical application of ADOs.

1. The basic assumption in many ADOs is a guaranteed upper and lower bound to the relative approximation error, whereas many applications only require a low average error and a low error variance. Although the latter is a weaker condition than the first, the latter condition is a better indicator of the overall accuracy of the oracle. This duality manifests itself in the fact that experimental approaches in the field of ADOs starting from guaranteed error bounds end up in reporting the average error (e.g. [2]). Qiao et al. [7] showed that the relaxation of these bounds results in a reduced complexity of the precalculation time, the oracle space and the query time.
2. Some applications require a minimal absolute approximation error, instead of a minimal relative one. This is the case when the transportation matrix for the traveling salesperson problem (TSP) is generated by an ADO. The stability regions [9] of the TSP edge lengths, i.e. the length domains within which the optimal sequence of nodes is identical, better fit an absolute than a relative deviation pattern.
3. Many popular applications of network distance approximation apply to transportation networks with time weights, supporting travel time estimation of the **'fastest'** path between two vertices. These spatial networks do not satisfy the triangle inequality, which is assumed to hold or applies to the evaluation network in most ADO research. The ADO for spatial networks in general by Sankaranarayanan and Samet [2] is solely validated by experiments on distance-weighted transportation networks. To the best of our knowledge, not

any ADO designed for spatial networks has been validated on time-weighted transportation networks before.

In order to evaluate an oracle based on the average error and its variance (concern 1), Section 2 formalizes the concept of an ADO minimizing the root-mean-square error (RMSE) of its network distance approximations. The rationale of this concept is to abandon the guarantee-based ADO design to increase the overall approximation accuracy (depending on the intrinsic characteristics of the network) or to reduce the space complexity of the oracle. Considering the second concern, this concept is introduced for both relative and absolute deviation. Section 3 comprises an ADO design for spatial networks minimizing the absolute RMSE. This oracle is constructed by partitioning the network into clusters of vertices that share similar paths of lowest cost. In Section 4, the oracle's quality is evaluated for a time-weighted transportation network extracted from the OpenStreetMap project.

2 Least squares approximate distance oracles

The following definitions apply to weighted graphs in general as well as to *directed* weighted graphs in general. In the latter type of graph $G = (V, E)$, any directed edge (or *arc*) $e \in E$ is defined as an *ordered* pair of vertices, corresponding to its associated direction. A path in a directed graph is a sequence of consecutive edges of forward direction.

2.1 Basic concepts

Given an ADO approximating a network distance query $d_G(u, v)$ by the *reported distance* $d_R(u, v)$ for any vertex u and v of a general network $G = (V, E)$. We define two inverse accuracy criteria, the absolute and the relative RMSE, as follows.

$$RMSE_{abs}(G) := \sqrt{\frac{\sum_{\forall u,v \in V, u \neq v} (d_G(u,v) - d_R(u,v))^2}{|V|^2 - |V|}}$$

$$RMSE_{rel}(G) := \sqrt{\frac{\sum_{\forall u,v \in V, u \neq v} \left[\frac{d_G(u,v) - d_R(u,v)}{d_G(u,v)}\right]^2}{|V|^2 - |V|}}$$

An oracle has an optimal approximation accuracy on a graph G, if the RMSE on G is minimal. These definitions however imply that the ADO accuracy evaluation requires an all-pairs shortest path approach, of which the processing time has a (nearly) cubic complexity: the time complexity of the classical Floyd-Warshall algorithm is $O(n^3)$; Chan's algorithm [10] requires $O(n^3 \cdot (log(log(n)))^3/(log(n))^2)$ time. Therefore, in practice, a representative query[1] sample set $S \subset V \times V$ is

[1] This query is an unordered/ordered pair, since $d_G(u, v)$ is a symmetric/asymmetric function in undirected/directed graphs.

determined. S has a random distribution in $V \times V$ or an expected query distribution for a certain application domain e.g. long-distance queries. The ADO's accuracy can be evaluated quickly based on the sampled RMSE on G as follows.

$$RMSE^S_{abs}(G) := \sqrt{\frac{\sum_{\forall(u,v)\in S}(d_G(u,v) - d_R(u,v))^2}{|S|}}$$

$$RMSE^S_{rel}(G) := \sqrt{\frac{\sum_{\forall(u,v)\in S}\left[\frac{d_G(u,v)-d_R(u,v)}{d_G(u,v)}\right]^2}{|S|}}$$

2.2 Oracles of unit size

We now introduce the optimal oracles of unit size for both the absolute and relative cases. These oracles are mainly intended as a reference test to compare the intrinsic characteristics of different networks, and their difficulty level of establishing an ADO. We expect for instance that the accuracy of a unit size oracle for spatial networks is considerably better on a distance-weighted than on a time-weighted transportation network. Their optimality is defined directly with regard to a query sample set $S \subset V \times V$, since an all-pairs shortest path calculation in fact realizes an exact distance oracle requiring a a high processing time complexity. Straightforward mathematical methods produce the following constant values.

Oracles of unit size in general networks. An ADO of size $O(1)$ minimizing $RMSE^S_{abs}(G)$, responds to each query $d_G(u,v)$ by a constant distance d_C:

$$d_C = \frac{\sum\limits_{\forall(u,v)\in S} d_G(u,v)}{|S|}$$

Analogously, it can be shown that an ADO of size $O(1)$ minimizing $RMSE^S_{rel}(G)$ and answering to each query a constant distance d_C requires

$$d_C = \frac{\sum\limits_{\forall(u,v)\in S}(1/d_G(u,v))}{\sum\limits_{\forall(u,v)\in S}(1/d^2_G(u,v))}$$

Oracles of unit size in spatial networks. The distortion for an individual vertex couple in a spatial network is defined as

$$\gamma(u,v) := \frac{d_G(u,v)}{d_S(u,v)}$$

Note that the maximum value of $\gamma(u,v)$ for any $u,v \in V$ is often referred to as the (maximum) distortion, the dilation or the stretch factor of G [2, 11].

Sankaranarayanan and Samet [2] state that a *distortion spectrum* usually exhibits large distortion values only for low spatial distances. An ADO of size $O(1)$ for spatial networks entails the approximation of a query $d_G(u, v)$ by the product $\gamma_C \cdot d_S(u, v)$, where γ_C represents a constant distortion. The following condition yields an oracle of optimal approximation accuracy with regard to the absolute RMSE.

$$\gamma_C = \frac{\sum\limits_{\forall(u,v)\in S} (d_S^2(u, v) \cdot \gamma(u, v))}{\sum\limits_{\forall(u,v)\in S} d_S^2(u, v)}$$

An optimal ADO of the same configuration minimizing the relative RMSE requires

$$\gamma_C = \frac{\sum\limits_{\forall(u,v)\in S} (1/\gamma(u, v))}{\sum\limits_{\forall(u,v)\in S} (1/\gamma^2(u, v))}$$

3 An advanced ADO based on clusters and transit nodes

The classical landmark embedding techniques based on spatial coverage, described in Section 1, are less effective in networks that do not necessarily satisfy the triangle inequality. The core of the ADO introduced in the present section is therefore based on graph partitioning into disjoint clusters of vertices. Furthermore, a set of *transit nodes* of minimal size is determined for each of the clusters. The network distance approximation algorithm is based on the precalculated distances between the transit nodes. The design of this oracle is aimed at minimizing the absolute RMSE. Related approaches to graph partitioning are discussed in the next paragraphs.

3.1 Related work

Several graph clustering algorithms have been proposed optimizing different objectives. Both Monien and Diekmann [12] and Pothen [13] minimize the number of edges connecting different partitions. Edge length based clustering of edges in graphs was proposed by Das and Narasimhan [14] for constructing sparse spanners in complete Euclidean networks. The Markov cluster algorithm by van Dongen [15] minimizes the probability of leaving the cluster during a random walk. It is an iterative algorithm on a matrix of transition probabilities. The conductance of a graph indicates the number of steps a random walk in the graph requires for converging to a uniform distribution. Clusters of low conductance can be seen as bottlenecks. Iterative Conductance Cutting [16, 17] maximizes the conductance of these bottlenecks.

In the following cases, graph partitioning has been applied in order to obtain the path of lowest cost in graphs. This discipline is somewhat different from network distance approximation because it mainly focusses on returning the complete path of the exact result, which implies a different trade-off between

space and query time complexity. Lansdowne and Robinson [18] were the first to apply the concept of spatial decomposition to the shortest path problem in sparse directed graphs. During query time they assign the vertices of the spatial graph to a set of regions in order to optimize the performance of the exact calculation of the n paths of lowest cost. More recent approaches divide the complete graph into clusters during the preprocessing phase. This phase also implies that the partitioning description is stored together with a set of precomputed paths or distances between the clusters. During a shortest path query, this stored information is used in order to drastically reduce the search space of the path calculation procedure. Huang et al. [19] advocate the application of the Spatial Partition Clustering technique in order to minimize I/O costs in routing systems that require to load the edges from secondary storage to a main memory buffer. It partitions the arcs of a directed spatial graph such that the origin vertices of the arc of a partition are bound by a quasi-square polygon. The routing algorithm by Flinsenberg et al. [20] only considers the edges (1) belonging to the cells of the start and destination vertex, (2) connecting two so-called *boundary nodes* of different cells, and, (3) representing precalculated paths between boundary nodes of same cells. A hierarchical version of this algorithm has been presented by Jung and Pramanik [21]. Flinsenberg et al. [20] introduced the partitioning problem as finding the cell configuration that yields a minimal average number of loaded edges. Their preprocessing phase consists of several runs, repeatedly merging 1-vertex-cells until one cell of size $|V|$ remains. This greedy merge procedure is managed by a priority function containing a random element. The cell configuration over all the runs that suits the partitioning problem best is selected. When, however, the A* algorithm is applied to a transportation network, Flinsenberg et al. [20] discovered that minimizing the number of loaded edges does not result in the fastest query results, and reformulated the partitioning problem's objective value minimizing the algorithm's search space. Maue et al. [22] assign each node to the cluster of the closest node (with regard to the network distance) of a set of k centre nodes. Their routing algorithm integrates pruning of complete clusters based on distance bounds. The resulting search space has the shape of a corridor around the shortest path, of which the narrowness is determined by the number of clusters k. Note that these approaches are different from recent successful partition-based approaches to exact shortest path calculation since the preprocessing phase starts from arbitrary graph partitions such as administrative divisions. In order to lower the number of precalculated paths, Bast et al. [23] search for the minimal set of *transit nodes* outside the partition, such that any long-distant shortest path from/to the partition passes one of these nodes.

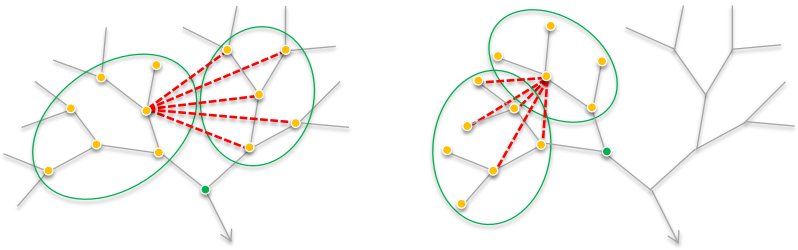
3.2 The oracle

Definitions. We first introduce a few definitions supporting the description of the advanced ADO. A random sample of $k < |S|$ elements of the set S is denoted $random(S, k)$. The *partition* of a set S is a collection of pair-wise disjoint subsets of this set such that the union of these subsets equals S. A directed graph $G(V, E)$

is *connected* when for each pair of nodes $(u, v) \in V \times V$ there exists a path from u to v. The *forward shortest path tree* of a connected directed graph $G(V, E)$ rooted at vertex $r \in V$ is a tree T in G, such that, for each node $v \in V$ the downward path from r to v in T corresponds with a path of lowest cost from r to v in G. The *backward shortest path tree* of a directed graph $G = (V, E)$ rooted at vertex $r \in V$ is a tree T in G, such that, for each node $v \in V$ the upward path from v to r in T corresponds with a path of lowest cost from v to r in G.

ADO construction. The construction of the advanced ADO from the connected directed spatial network $G(V, E)$ comprises stepwise generation of the following elements:

1. two partitions of V, of which P_O consists of *origin clusters* and P_D of *destination clusters*,
2. a set $T \subset V$ of minimal size, containing transit nodes for each cluster C in P_O and P_D (we say $t \in T$ is a transit node of C),
3. the precalculated distances $d_G(s, t)$ for any transit node s of an origin cluster and any transit node t of a destination cluster (close cluster pairs excluded, see element 6),
4. the precalculated distances $d_G(u, s)$ between any u in an origin cluster C_O and any transit node s of C_O,
5. the precalculated distances $d_G(t, v)$ between any transit node t of a destination cluster C_D and any v in C_D,
6. a set of close cluster pairs $(C_O, C_D) \in P_O \times P_D$, and,
7. the constant distortions γ_{C_O,C_D} for any close cluster pair (C_O, C_D).



(a) (b)

Fig. 1: Association graph construction. The subfigures show two consecutive runs of the recursive algorithm. The backward shortest path tree is indicated by a solid line. The association edge creation or counter increment (dashed lines) is only illustrated for one vertex. Ellipses delimit the subtrees with $maxDepth = 2$. Note that any pair of vertices is linked at most once during the processing of one tree.

Procedure `GenerateOriginPartition(`G`)`
input : $G \leftarrow$ spatial network $G(V, E)$
output : $P_O \leftarrow$ set of subsets of V
parameters: $numberOfRoots$, $popularityThreshold$

▷ *construction of sample backward shortest path trees*
roots ← `random(`$V, numberOfRoots$`)`
trees ← tree set (initially empty)
foreach r *in* roots **do**
  tree ← `backwardShortestPathTree(`G,r`)`
  `insert(`trees, tree`)`

▷ *association graph construction*
$G_a \leftarrow$ association graph $G_a(V_a, E_a)$ (initially empty)
▷ *an association graph is an undirected graph where each edge e has a weight* $\mathsf{w}(e)$ *and a counter* $\mathsf{c}(e)$*, and each vertex v has a vote* $\mathsf{v}(v)$*, all equal to 0 by default*
foreach t *in* trees **do**
  $G_a \leftarrow$ `generateAssociations(`G_a, t, `root(`t`))`

▷ *association edge weight calculation*
foreach *edge* e_a *in* E_a **do**
  $\{v_1, v_2\} \leftarrow$ `getVertices(`G_a, e_a`)`
  foreach e_1 *in* `getEdges(`G_a, v_1`)` **do**
    foreach e_2 *in* `getEdges(`G_a, v_2`)` **do**
      if *(*`getVertices(`G_a, e_1`)` $\bigcap$ `getVertices(`G_a, e_2`)` *)* $\setminus \{v_1, v_2\} \neq \emptyset$
      then $w(e_a) \leftarrow w(e_a) + c(e_1) \cdot c(e_2)$

▷ *determination of principal nodes*
foreach *vertex* v_a *in* V_a **do**
  $\{v_h\} \leftarrow$ `getVertices(`G_a, `edgeMaxWeight(getEdges(`G_a, v_a`)))` $\setminus \{v_a\}$
  $\mathsf{v}(v_h) \leftarrow \mathsf{v}(v_h) + 1$
principalNodes ← a subset of V_a (initially empty)
foreach *vertex* v_a *in* V_a **do**
  popularity ← $\mathsf{v}(v_a)/|$`getEdges(`G_a, v_a`)`$|$
  if popularity $>$ $popularityThreshold$ **then** `insert(`principalNodes, v_a`)`

▷ *cluster construction*
$P_O \leftarrow$ set of singletons of elements in principalNodes
foreach *vertex* v_a *in* $V_a \setminus$principalNodes **do**
  $E_p \leftarrow$ subset of `getEdges(`G_a, v_a`)` connecting to a vertex in principalNodes
  if $E_p \neq \emptyset$ **then**
    $\{v_h\} \leftarrow$ `getVertices(`G_a, `edgeMaxWeight(`E_p`))` $\setminus \{v_a\}$
    `addToSubsetContaining(`P_O, v_a, v_h`)`
while $V_a \setminus (\bigcup_{C \in P_O} C) \neq \emptyset$ **do**
  foreach *vertex* v_a *in* $V_a \setminus (\bigcup_{C \in P_O} C)$ **do**
    $E_c \leftarrow$ subset of `getEdges(`G_a, v_a`)` connecting to a vertex in $\bigcup_{C \in P_O} C$
    if $E_c \neq \emptyset$ **then**
      $\{v_h\} \leftarrow$ `getVertices(`G_a, `edgeMaxWeight(`E_c`))` $\setminus \{v_a\}$
      `addToSubsetContaining(`P_O, v_a, v_h`)`
return P_O

Algorithm 1: Generation of the origin cluster partition. The set, tree and graph functions are explained in Table 1.

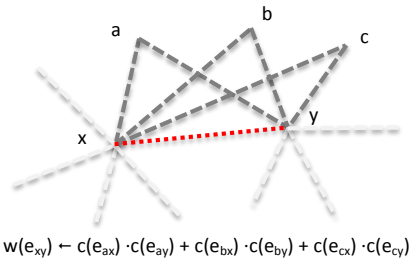


Fig. 2: Association edge weight calculation example. The weight of the edge between vertices x and y is calculated as the sum of the counter products of the edges connecting x and y with a common vertex.

	Function	Definition
Sets	$random(S, k)$	returns a random sample of $k < \|S\|$ elements of the set S
	$insert(S, e)$	inserts e in the set S
	$addToSubsetContaining(S, e_{add}, e_{ref})$	inserts e_{add} in the subset of S containing e_{ref}
Trees	$root(T)$	returns the root of tree T
	$depth(e, T)$	returns the depth of element e in tree T (the root of T has depth 0)
	$next(e, T)$	returns the set of children of element e in tree T
	$getSubtrees(e, T, d)$	returns the set of trees rooted at the child elements of e in tree T and chopped of at depth d relative to the child elements
Graphs	$getVertices(G, e)$	returns the set of 2 vertices connected by edge e in graph G
	$getEdges(G, v)$	returns the set of edges connecting vertex v in graph G
	$edgeMaxWeight(E)$	returns the edge e that has the highest weight $w(e)$ in edge set E
	$connected(G, v_1, v_2)$	there exists an edge between v_1 and v_2 in graph G
	$constructEdge(G, v_1, v_2)$	inserts an edge between v_1 and v_2 in graph G
	$backwardShortestPathTree(G, r)$	returns the backward shortest path tree (defined in the definitions paragraph of Section 3) of graph G rooted at vertex r

Table 1: Set, tree and graph functions

```
Procedure generateAssociations(G_a, t, el)
    input      : G_a ← association graph G_a(V_a, E_a)
    input      : t ← tree of elements in V_a and branches in E_a
    input      : el ← an element in t
    output     : G_a ← association graph G_a(V_a, E_a)
    parameters: minDepth, maxDepth
    if depth(el, t) > minDepth then
        subtrees ← getSubtrees (el, t, maxDepth)
        foreach 2-element combination (tree_1, tree_2) in subtrees do
            foreach element el_x in tree_1 do
                foreach element el_y in tree_2 do
                    if not(connected(G_a, el_x, el_y)) then
                        constructEdge(G_a, el_x, el_y)
                    c(getEdge(G_a, el_x, el_y)) ← c(getEdge(G_a, el_x, el_y)) +1
    foreach element el_n in next(el, t) do
        G_a ← generateAssociations(G_a, t, el_n)
    return G_a
```

Algorithm 2: Recursive generation of the association graph. The set, tree and graph functions are explained in Table 1.

Element 1 implies clustering of vertices sharing many similar paths in the set of all paths of lowest cost starting (partition P_O) or ending (partition P_D) in these vertices. This similarity can be described best in terms of the LCA of two vertices in a forward/backward shortest path tree. Two nodes are similar when their LCA is only a few edges away for many paths of lowest cost. Algorithm 1 describes a sampling-based method of low computational time to generate the origin cluster partition P_O. It generates an association graph G_a from a set of $numberOfRoots$ backward shortest path trees in G rooted in a sample of V. This undirected graph registers in its edge counters how many times two vertices have a close LCA, over the set of trees. This registration (Algorithm 2) is realised for one tree t by linking the vertices between any pair of subtrees, cropped at depth $maxDepth$, of any vertex located at least at depth $minDepth$ in t. This is shown in Figure 1. The edge weight calculation propagates the edge counters over triangle subgraphs of G_a, as illustred in Figure 2. Principal nodes are the vertices of V_a around which the clusters will be built. Principal node determination starts by a voting mechanism. Any vertex in V_a submits a vote for the vertex connected by the edge of heaviest weight. Next, vertices that have a vote-degree ratio above the $popularityThreshold$ parameter, become the principal nodes. Any vertex connected in G_a to a principal node is assigned to the cluster of the principal node connected to the vertex by the edge of heaviest weight. Finally, the other vertices are iteratively added to the clusters corresponding to their edge of heaviest weight. The algorithm generating the destination cluster partition P_D is the same but starts from a set of forward shortest path trees. Note that both types of shortest path tree can be generated by the Dijkstra algorithm. In case of a backward shortest path tree, it is required

to invert any of the graph's edge directions before calculation, and to interpret the results accordingly.
For element 2, a minimal set of transit nodes for any cluster is determined. In case of origin/destination clusters, the paths in the sample backward/forward shortest path trees starting/ending in any vertex of the cluster are considered. The coverage of the transit nodes of a cluster is the proportion of these paths that passes at least one of the transit nodes. The minimal set of transit nodes for any cluster is determined, which has at least a specified coverage $tnCoverage$ (e.g. 95%). This minimal set is retrieved through a mixed integer programming approach. The precalculated distances in the elements 3, 4 and 5 are calculated using a one-to-many shortest path algorithm. While the above distances are used to approximate long network distance queries, the probability that short paths of lowest cost pass through the transit nodes is remarkably lower. For element 6, a set of close cluster pairs is determined. Two clusters are close if they have at least one vertex in common. For queries from cluster C_O to C_D, the network distance approximation will not be based on the precalculated distances, but on the constant distortion minimizing the absolute RMSE for queries in $C_O \times C_D$. These distortions (of the last element) are calculated for a sample query set $random(C_O \times C_D, k_{avdist})$.

Network distance approximation algorithm. During a query for the network distance $d_G(u, v)$, the origin cluster C_O and destination cluster C_D are retrieved where $u \in C_O$ and $v \in C_D$. When (C_O, C_D) is not a close cluster pair, $d_R(u, v)$ equals the minimal value of $d_G(u, s) + d_G(s, t) + d_G(t, v)$ for any combination of transit node s of C_O and transit node t of C_D. Otherwise, $d_R(u, v)$ is $\gamma_{C_O,C_D} \cdot d_S(u, v)$.

Complexity. c denotes the average number of partitions in P_O and P_D, t denotes the average number of transit nodes per cluster, and l denotes the average number of clusters in P_D that forms a close cluster with a cluster in P_O. Suppose that any of the 7 elements of the advanced ADO is stored in a hash table of space complexity $O(k)$ and average time complexity $O(1)$. The space complexity built up by these elements is: $O(2{\cdot}n{+}2{\cdot}c{\cdot}t{+}c{\cdot}(c{-}l){\cdot}(t)^2{+}n{\cdot}t{+}n{\cdot}t{+}c{\cdot}l{+}c{\cdot}l) = O(n{+}c^2)$, assuming that l and t are much smaller than c and n. This means that the space complexity is linear with regard to the number of nodes and quadratic with regard to the number of clusters. The query time complexity is $O(1)$ in case the origin and destination are located in a close cluster pair. Otherwise, it is $O(t^2)$.

4 Experiment

The advanced ADO introduced in the Section 3 is evaluated on a time-weighted transportation network extracted from OpenStreetMap, which is a source of publicly available geographic data. Mapping individual *way* objects in an OpenStreetMap map extract to a directed weighted spatial graph is described in Appendix A. After this extraction, a minimal number of vertices is removed

from the graph such that the graph becomes connected. This process starts by the manual selection of a vertex r which is known to be in the largest connected subgraph. Next, both a forward and a backward shortest path tree rooted in r is constructed. The subgraph consisting of the nodes belonging to both trees is a connected graph.

The *Ghent dataset* is the OpenStreetMap map extract of latitude range [51.0258, 51.0834] and longitude range [3.6685, 3.7856], dated *May* 16, 2012. The derived connected graph contains 4296 vertices.

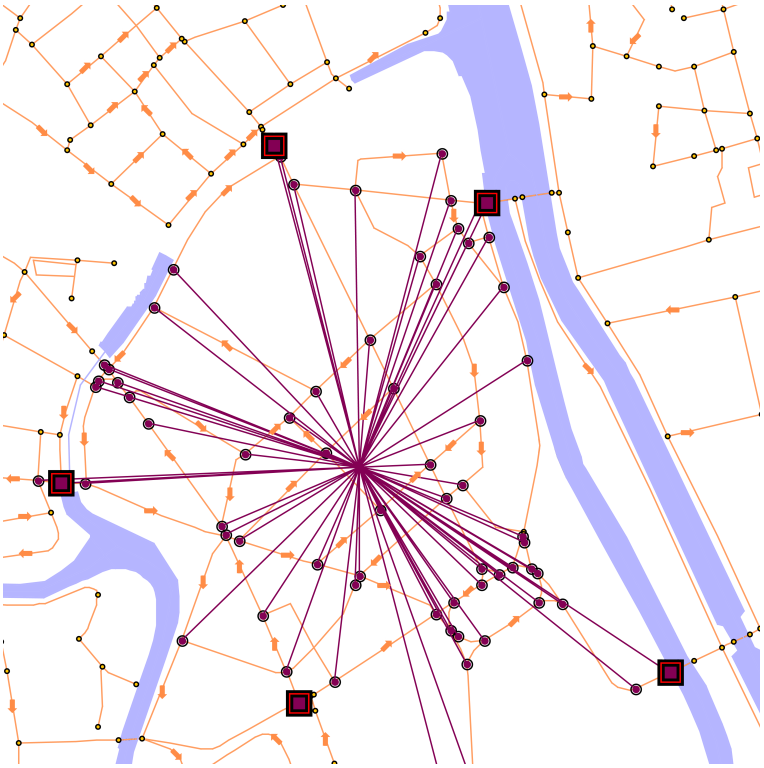


Fig. 3: Instance of an origin cluster in the Ghent dataset. All vertices belonging to the cluster are connected by a star (trivial centre point). The other lines indicate the edges of the transportation network. Arrows indicate the major part of edges that are traversable in single direction. The 5 boxes represent the cluster's associated transit nodes, covering at least 90% of the sample paths leaving the cluster. Network and waterways derived from © OpenStreetMap contributors.

The basic evaluation for this directed weighted spatial graph $G(V, E)$ assumes construction of two independent sets S and T of random ordered pairs

(queries) in $V \times V$. Both sets contain 10000 elements. The positions of the individual vertices in the pairs of S and T have a uniform distribution over the rectangular area of interest. We assume that this is the expected query distribution for main-purpose applications of ADOs. Next, an ADO of unit size minimizing $RMSE^{T}_{abs}(G)$, and the advanced ADO introduced in Section 3 are constructed. The construction parameter settings for the latter oracle were $numberOfRoots = 50$, $popularityThreshold = 0.07$, $minDepth = 15$, $maxDepth = 10$, $tnCoverage = 0.90$, $k_{avdist} = 20$. These settings resulted in 87 origin and 79 destination clusters. One of the origin clusters, generated by this oracle, is shown together with its associated transit nodes in Figure 3. Table 2

			percentile						
	RMSEabs	*avg*	*1%*	*5%*	*10%*	*90%*	*95%*	*99%*	*max*
Unit size ADO	2.814	2.207	0.03	0.17	0.33	4.64	5.60	7.42	13.60
Advanced ADO	0.777	0.253	0.00	0.00	0.00	0.90	1.79	3.63	9.01

Table 2: Absolute error statistics (in minutes) of network distance approximations for the query sample set S in the Ghent dataset.

shows the absolute error $RMSE^{S}_{abs}(G)$ and some other absolute error statistics for both oracles.

In order to analyse the oracle's approximation accuracy for different categories of "as the crow flies" distance between the origin and destination vertex of a query, we introduce the set U of 10000 random queries (u, v) in $V \times V$ where $d_S(u, v)$ has a uniform distribution. Both $RMSE^{U}_{abs}(G)$ and the average absolute error for U were found to be lower sc. 0.730 and 0.211. Figure 4 shows the unit size oracle's absolute approximation error as a function of the query's spatial distribution for any query in the sample set of universal spatial distance distribution U. This data is averaged for discrete spatial distance ranges in the histogram of Figure 6. The same data on the advanced oracle's approximation accuracy is represented in Figure 5 and 7. The first histogram shows that the unit size oracle's absolute error average is around 2 minutes for medium-distant queries, but increases for longer distances up to 8 minutes. The other oracle's absolute error average is close to 0 minutes for long-distant queries, shows a peak of about 1.2 minutes around queries of distance 500m. This shows that the latter oracle is able to drastically reduce the absolute error and that it is the most susceptible to absolute errors when the origin and destination vertices are located in a close cluster pair. While the approximation mechanism for non-close cluster pairs yields a reasonable chance to have an exact approximation, the mechanism for close cluster pairs is based on spatial distance multiplication by a constant. The latter mechanism can be seen as an improved version of the unit size oracle. For queries of length 400-500m, its approximations error is the halve of the one of the unit size oracle.

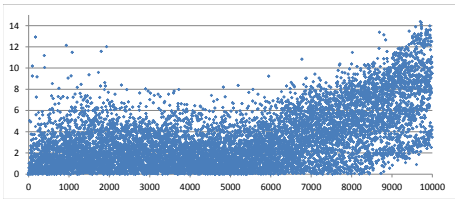


Fig. 4: Unit size oracle's absolute error as a function of spatial distance.

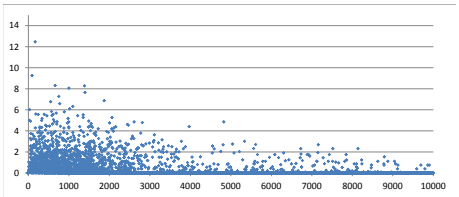


Fig. 5: Advanced oracle's absolute error as a function of spatial distance.

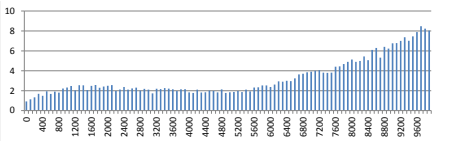


Fig. 6: Unit size oracle's average absolute error histogram for spatial distance ranges.

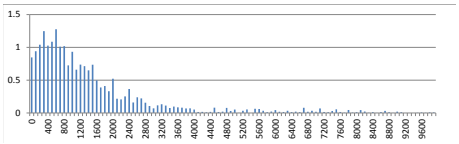


Fig. 7: Advanced oracle's average absolute error histogram for spatial distance ranges.

5 Conclusion

The present paper introduces a framework for the evaluation of ADOs based on the root-mean-square error (RMSE) of the absolute or relative error of a sample set of network distance approximations in a graph $G(V, E)$. This framework applies both to general and spatial networks. An RMSE minimizing unit size oracle for general networks generates a constant approximation. The version for spatial networks approximates the network distance by the product of the spatial distance and a constant distortion.

An advanced ADO was introduced. It organizes the vertices V into a partition of origin clusters and one of destination clusters. Each cluster is assigned a set of transit nodes. A close cluster pair is a couple of an origin and a destination cluster sharing at least one vertex. Network distance approximation between vertices of a close cluster pair is based on the constant distortion minimizing the absolute RMSE for a sample set of queries from the first to the second cluster of the pair. Approximation between remote vertices is based on a set of precalculated distances between the vertices and the transit nodes and between transit nodes of different clusters. The oracle's average space complexity is $O(n + c^2)$ and its average query time complexity is $O(t^2)$, where c is the average number of clusters in a partition and t the average number of transit nodes of a cluster. Its prediction accuracy was evaluated in terms of the RMSE of the absolute error of the reported distances. The comparison with a unit size oracle for spatial networks was made for the Ghent dataset for a query sample set of size 10000. The positions associated with the individual vertices in this query sample set are uniformly distributed over the rectangular dataset area. The advanced ADO

realizes a reduction of 3.5 times the RMSE and of 8.5 times the average absolute error, in comparison with the unit size oracle.

Acknowledgment

The research has been carried out partly in the context of the industrial PhD project "Structural heuristics for personalized routes" funded by the IWT (090726) and the company RouteYou.

References

1. Thorup, M., Zwick, U.: Approximate distance oracles. In Vitter, J.S., Spirakis, P.G., Yannakakis, M., eds.: Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece, ACM (2001) 183–192
2. Sankaranarayanan, J., Samet, H.: Distance oracles for spatial networks. In: Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on. (2009) 652–663
3. Laporte, G.: The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research **59**(3) (1992) 345–358
4. Vansteenwegen, P., Souffriau, W., Van Oudheusden, D.: The orienteering problem: A survey. European Journal of Operational Research **209**(1) (2011) 1 – 10
5. Sommer, C., Verbin, E., Yu, W.: Distance oracles for sparse graphs. In: 50th IEEE Symposium on Foundations of Computer Science (FOCS). (2009) 703–712
6. Gudmundsson, J., Levcopoulos, C., Narasimhan, G., Smid, M.: Approximate distance oracles for geometric spanners. ACM Trans. Algorithms **4**(1) (March 2008) 10:1–10:34
7. Qiao, M., Cheng, H., Yu, J.X.: Querying shortest path distance with bounded errors in large graphs. In: Proceedings of the 23rd international conference on Scientific and statistical database management. SSDBM'11, Berlin, Heidelberg, Springer-Verlag (2011) 255–273
8. Qiao, M., Cheng, H., Chang, L., Yu, J.X.: Approximate shortest distance computing: A query-dependent local landmark scheme. In Kementsietsidis, A., Salles, M.A.V., eds.: IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012, IEEE Computer Society (2012) 462–473
9. Libura, M., van der Poort, E.S., Sierksma, G., van der Veen, J.A.: Stability aspects of the traveling salesman problem based on k-best solutions. Discrete Applied Mathematics **87**(13) (1998) 159 – 185
10. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing. STOC '07, New York, NY, USA, ACM (2007) 590–598
11. Narasimhan, G., Smid, M.: Approximating the stretch factor of euclidean graphs. SIAM J. Comput. **30**(3) (May 2000) 978–989
12. Monien, B., Diekmann, R.: A local graph partitioning heuristic meeting bisection bounds. In: Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, PPSC 1997, March 14-17, 1997, Hyatt Regency Minneapolis on Nicollel Mall Hotel, Minneapolis, Minnesota, USA, SIAM (1997)

13. Pothen, A.: Graph partitioning algorithms with applications to scientific computing. In: Parallel Numerical Algorithms, Kluwer Academic Press (1997) 323–368
14. Das, G., Narasimhan, G.: A fast algorithm for constructing sparse euclidean spanners. Int. J. Comput. Geometry Appl. **7**(4) (1997) 297–315
15. van Dongen, S.M.: Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht, The Netherlands (2000)
16. Kannan, R., Vempala, S., Veta, A.: On clusterings-good, bad and spectral. In: Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on. (2000) 367–377
17. Brandes, U., Gaertler, M., Wagner, D.: Experiments on graph clustering algorithms. In Battista, G., Zwick, U., eds.: Algorithms - ESA 2003. Volume 2832 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2003) 568–579
18. Lansdowne, Z.F., Robinson, D.W.: Geographic decomposition of the shortest path problem, with an application to the traffic assignment problem. Management Science **28**(12) (1982) 1380–1390
19. Huang, Y.W., Jing, N., Rundensteiner, E.A.: Optimizing path query performance: graph clustering strategies. Transportation Research Part C: Emerging Technologies **8**(16) (2000) 381 – 408
20. Flinsenberg, I.I., van der Horst, M.M., Lukkien, J.J., Verriet, J.J.: Creating graph partitions for fast optimum route planning. WSEAS Transactions on Computers **3**(3) (2004) 569 – 574
21. Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. Knowledge and Data Engineering, IEEE Transactions on **14**(5) (2002) 1029–1046
22. Maue, J., Sanders, P., Matijevic, D.: Goal-directed shortest-path queries using precomputed cluster distances. J. Exp. Algorithmics **14** (January 2010) 2:3.2–2:3.27
23. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast routing in road networks with transit nodes. Science **316**(5824) (2007) 566

A Extraction of a directed weighted spatial graph from OpenStreetMap data

The following rules describe a mapping of OpenStreetMap (OSM) data to a car transportation network, in the form of a directed weighted spatial graph $G(V, E)$.

An OSM *way* instance is **traversable** if

- it does not have a *visible*-tag of value *false*,
- it does have a *highway*-tag, and,
- the value of this tag is different from *bridleway*, *bus_guideway*, *construction*, *cycleway*, *footway*, *path*, *pedestrian*, *proposed*, *raceway*, *service* or *steps*.

If a *way* instance has a *oneway*-tag, this *way* is traversable

- in the **forward direction** if this tag's value equals *yes*, *true* or *1*,
- in the **backward direction** if this tag's value equals *-1*,
- in **both directions** if this tag's value equals *no*, *false* or *0*.

Otherwise, if this *way* instance has a *junction*-tag, this *way* is traversable in the **forward direction** if this tag's value equals *roundabout*.
Otherwise, if the *highway*-tag's value of this *way* equals *motorway_link* this *way* is traversable in the **forward direction**.
In any other case this way is traversable in **both directions**.

If a way instance has a *maxspeed*-tag, its **speed** in km/h is defined as:

- 130 if this tag's value equals *none*,
- 50 if this tag's value equals *signals*,
- this tag's value if it is a number,
- this tag's value multiplied by 1.609344 if it is a number followed by *mph*.

Otherwise, when the way instance has a *maxspeed*-tag and its value has the *countrycode:waytype* pattern, the country code and way type yield a **speed** according to the following mapping.
(((BE, motorway), 120), ((BE, trunk), 90), ((BE, primary), 90), ((BE, secondary), 90), ((BE, tertiary), 90), ((BE, residential), 30), ((BE, living_street), 20),
((NL, motorway), 120), ((NL, trunk), 100), ((NL, primary), 80), ((NL, secondary), 80), ((NL, tertiary), 80), ((NL, living_street), 15),
((ES, motorway), 120), ((ES, trunk), 100), ((ES, primary), 90), ((ES, secondary), 90), ((ES, tertiary), 90), ((ES, residential), 30), ((ES, living_street), 20))
Otherwise, the same mapping applies to the country code derived from the data extract's metadata and the value of the *highway*-tag.
When the mapping failed, its **speed** is 50.

We define the **reduced node reference list** of a traversable *way* instance as the list of node references appearing as the first or the last element in the *node reference list* of any traversable *way* instance. The **waylength** between the nodes n_i and n_j appearing in a full *node reference list* of a *way* instance equals the sum of the spherical distances in metres between any two consecutive nodes n_k and n_{k+1} in the full *node reference list*, where $i \leq k < j$.

Any OSM *node* instance appearing in the reduced node reference list of a traversable *way* instance maps to a vertex of V. The values of its *lon*-tag and *lat*-tag map to the spatial position in $\mathbb{R}^2$ of the vertex. $d_S(u, v)$ indicates the spherical distance in metres between the vertices u and v.
Any two consecutive nodes of the reduced node reference list of a *way* traversable in the forward or in both directions maps to a directed edge of E starting in the first corresponding vertex and ending in the second corresponding vertex.
Any two consecutive nodes of the reduced node reference list of a *way* traversable in the backward or in both directions maps to a directed edge of E starting in the second corresponding vertex and ending in the first corresponding vertex.
The weight of an edge e from vertex u to v corresponds to $\frac{waylength(u,v) \cdot 0.06}{speed(e)} +$

0.167, where $speed(e)$ refers to the speed of the *way* instance from which e is generated.