A Hybrid Shortest Path Algorithm for Intra-Regional Queries on Hierarchical Networks

Gutemberg Guerra-Filho[1] and Hanan Samet[2*]

[1] Department of Computer Science and Engineering,
University of Texas at Arlington,
Arlington, TX USA
[2] Department of Computer Science,
University of Maryland, College Park

Abstract. A hierarchical approach to the single-pair shortest path problem subdivides a network with n vertices into r regions with the same number m of vertices ($n = rm$) and iteratively creates higher levels of a hierarchical network by merging a constant number c of adjacent regions. In a hierarchical approach, shortest paths are computed at higher levels and expanded towards lower levels through intra-regional queries. We introduce a hybrid shortest path algorithm to perform intra-regional queries. This strategy uses a subsequence of pre-processed vertices that belong to the shortest path while actually computing the whole shortest path. At the lowest level, the hybrid algorithm requires $O(m)$ time and space assuming a uniform distribution of vertices. For higher levels, the path view approach takes $O(1)$ time and requires $O(c^k m)$ space.

1 Introduction

Information systems that assist drivers in planning a travel are required to improve safety and efficiency of automobile travel. These systems use real-time traffic information gathered by traffic control and surveillance centers like traffic congestion and roadwork. They aid travelers in finding the optimal path to their destinations considering distance, time and other criteria. This helps to eliminate unnecessary travel time reducing accidents and pollution.

A *Moving Object Database* (MOD) is a special form of a spatial database that represents information about moving objects, their location in space, and their proximity to other entities or objects. In spatial database applications, the embedding space consists in a geographical network with a distance metric based on shortest paths. Hence, shortest path finding is a basic operation in MODs. This operation is used as a subroutine by many other proximity queries, including nearest neighbors [10]. In particular, finding nearest neighbors in a spatial network presumes that the shortest path to the neighbors have been computed already. The efficiency issues related to this operation are critical to

* This work was supported in part by the National Science Foundation under Grants IIS-10-18475, IIS-09-48548, IIS-08-12377, and CCF-08-30618.

MODs due to the dynamic and real-time characteristics of such databases. A large number of queries in a huge network may prevent the system from meeting the real-time requirement when a non-efficient approach is used.

A *path view* or *transitive closure* contains the information required to retrieve a shortest path corresponding to each pair of vertices in the network. This strategy pre-computes all shortest paths in the network. Once the path view is created, any path query is performed with a lookup in the path view and reporting the sequence of vertices that represent the path. A path query takes $O(n)$ time using a path view, since the number of vertices in the path may be linear in the worst case. The path view strategy requires $O(n^2)$ time for pre-processing [5, 8] and $O(n^2)$ space. The quadratic space is achieved when a predecessor matrix represents the path view. Methods that pre-compute and store the shortest paths between every pair of vertices in a graph assume that the real-time computation of the shortest paths for large networks may not be feasible. However, the focus of their work is the compact encoding of the $O(n^2)$ shortest paths and efficient retrieval. However, a major drawback of path view approaches is that large networks may need an unacceptable amount of time and space in order to satisfy the real-time constraint.

A hierarchical approach for shortest path finding is one possible way to satisfy both time and space efficiency requirements for moving object databases based on large geographical networks. A *hierarchical approach* subdivides a single network into r smaller regions. Each such region has the same number m of vertices and belongs to the lowest level of the hierarchy. The same size for regions is a property enforced at the other levels by creating a higher level merging c adjacent regions. This process creates a hierarchy of multilevel networks for a path finding search. The parameters $\langle r, m, c\rangle$ completely define the structure of a simple hierarchy of networks.

Given a hierarchy of networks, a hierarchical shortest path algorithm starts at the lowest level network from the source vertex. When the current region is completely traversed, the search is promoted to the next higher level. The promotion step is executed once at each level until it reaches the highest level. At this point, the search is demoted to the next lower level towards the destination vertex. The demotion step is performed until encountering the lowest level network. The resulting *hierarchical shortest path* consists of subpaths at each level of the network. If we seek only the shortest path cost, the hierarchical path cost is enough. However, when the actual path is required, the hierarchical approach executes shortest path queries inside individual regions, denoted here as *intra-regional queries*, for each level in order to expand these subpaths to the next lower level. These intra-regional queries are performed until the whole path is represented by subpaths at the lowest level network.

In a hierarchical approach, the shortest subpaths found at higher levels are expanded towards lower levels through intra-regional queries. Formally, an intra-regional shortest path query concerns the computation of a shortest path inside one region between two vertices on the border of the region. Therefore, these intra-regional queries are an essential component of a hierarchical approach for the computation of shortest paths in geographical networks. The subdivision of large geographical networks is not restricted to hierarchical approaches and, consequently, intra-regional queries are used in other frameworks. As an example, the TIGER files of the U.S. Census Bureau are subdivided into counties.

In this paper, we propose a new strategy to execute intra-regional shortest path queries at the lowest level of a hierarchical network. This strategy is based on a hybrid shortest path algorithm that uses both a partial path view and a multiple-source shortest path algorithm. A partial path is a pre-computed subsequence of vertices that belong to the shortest path. The partial path is used to find a single shortest path between two vertices on the border of a region. The novel contribution of our paper is this new strategy for intra-regional shortest path queries.

Using our hybrid shortest path algorithm, the time and space requirements for intra-regional queries at the lowest level of the hierarchical network is $O(m)$ assuming a uniform distribution of vertices. This is an improvement compared to the $O(m^{1.5})$ space required by a path view algorithm. The time efficiency is achieved by extending a shortest path algorithm to consider pre-computed guide vertices and to visit a much smaller number of vertices in the process.

The rest of this paper is organized as follows. In Section 2, we review work related to intra-regional queries. Hierarchical approaches are described in Section 3. We present our hybrid shortest path algorithm in Section 4. Section 5 presents our experimental results. Section 6 discusses the main contributions of this paper.

2 Related Work

This work is a natural progression from our prior work on building systems to support both feature-based queries ("Where is X happening?") and location-based queries ("What is happening at location Y?") [1] as in systems such as QUILT [20] and the SAND Browser [12]. It is also applicable to surface data (*e.g.*, [22]). Queries on road networks have received particular interest [18] with shortest path finding receiving renewed attention due to applications in spatial network databases such as MapQuest, Google Maps, Yahoo! Maps, Bing Maps, and others. Among these applications, nearest neighbors algorithms [2, 3] require the efficient computation of shortest path distances in spatial networks. However, besides these client applications, shortest path finding has been recently addressed with regards to the efficient encoding of path views. Samet *et al.* [13–15] pre-compute the shortest paths between all possible vertices in the network. The path view is encoded by subdividing the shortest paths from a single vertex based on the first edges of each shortest path. They further reduce the space requirements to store the path view by exploring spatial coherence with a shortest path quadtree. Similarly, Sankaranarayanan *et al.* [16, 17, 19] propose a new encoding of the path view that aggregates source and destination vertices into groups that share common vertices or edges on the shortest paths between them.

Frederickson [5] proposed a hierarchical algorithm for the single-source shortest path problem on planar graphs. The shortest paths between every pair of border vertices are found for two levels. Therefore, the algorithm uses a path view in the search for a shortest path through the hierarchy of networks.

Jing *et al.* [9] suggested a path view approach that stores the direct successor vertex and the cost of a shortest path for each source-destination pair in a region. Therefore, the path view requires $O(m^2)$ space for a region at the lowest level, where m is the number of vertices in a single region. They use a path finding algorithm that recursively queries the shortest path cost through all levels in the hierarchy of networks. They first determine the sets B_s and B_d of border vertices in regions containing the source vertex s and the destination vertex d, respectively. The algorithm uses pre-computed shortest paths between s and B_s; B_s and B_d; and B_d and d to find the global minimum cost for the path from s to d by searching among all pairs (b_s, b_d) of border vertices, where $b_s \in B_s$ and $b_d \in B_d$. The algorithm does not compute the whole path described by edges at the lowest level.

Shekhar *et al.* [21] focus on path view implementations for a two-level hierarchy. They proposed a hybrid path view that encodes the direct successor and cost for any shortest path only from interior vertices to the border vertices in each lowest level region. The higher level is fully materialized. Grid graphs were used in a complexity analysis of the space required for the path views. The space storage required for the path views is $O(n^{5/3})$. In this paper, we present a hybrid path view that requires $O(n)$ space based on partial paths.

Goldberg and Harrelson [6] propose a flat shortest path algorithm that prunes the number of visited vertices as does our hybrid shortest path algorithm for the lowest level of the hierarchy. They use A^* search with cost bounds computed according to the triangle inequality and distances between sampled (possibly random) vertices called landmarks.

3 A Hierarchical Approach

A *hierarchical approach* is based on the subdivision of the original network into regions. A region corresponds to a connected subgraph of the graph representing the network. A higher level network consists only of border vertices. A *border vertex* is a vertex that belongs to at least two different regions in the network. Since a shortest path passing through more than one region must include border vertices, the edges of a higher network represent possible connections between border vertices in this network.

The 0-level network is the original network represented by an embedding of an undirected planar graph $G(V, E)$ on the plane, where V (E) is the set of vertices (edges) in G. The number of vertices in V is n. This graph is subdivided into r smaller connected regions corresponding to subgraphs $\langle G_0^0, G_0^1, \ldots, G_0^{r-1}\rangle$ such that these subgraphs cover the original network ($V = V_0^0 \cup V_0^1 \cup \cdots \cup V_0^{r-1}$, $E = E_0^0 \cup E_0^1 \cup \cdots \cup E_0^{r-1}$) and each edge belongs to only one subgraph.

Each 0-level subgraph has m vertices and forms a suitable subdivision of the graph G where boundaries of each corresponding region has a size of $O(\sqrt{m})$ vertices. Such suitable subdivision is obtained in $O(n \log n)$ time using a fragmentation algorithm [5]. Goodrich [7] proposed an algorithm to find a separator decomposition and a suitable subdivision in $O(n)$ time.

For $k > 0$, the k-level network is generated from the $(k-1)$-level network. A vertex v is in the k-level network if it belongs to two different $(k-1)$-level regions. Note that the k-level network has only border vertices. There is an edge connecting two k-level vertices in the k-level network if there is a path connecting them in the same $(k-1)$-level region. The k-level network is subdivided into connected k-level regions containing c adjacent $(k-1)$-level regions such that each $(k-1)$-level region belongs to only one k-level region.

In a hierarchy of networks, the graph G associated with the original network is represented by a set G_0 of r subgraphs $\langle G_0^0, G_0^1, \ldots, G_0^{r-1} \rangle$. These subgraphs correspond to regions that will be merged into higher level regions containing c regions[3]. This way, all regions at a particular level have about the same number of vertices. Each subgraph is denoted by $G_k^i \left(V_k^i, E_k^i\right)$, where k is the level in the hierarchy and i represents a region. The set of subgraphs representing regions for a k-level is denoted by G_k. The set of border vertices in a subgraph G_k^i is represented by B_k^i and B_k denotes the border vertex set for the k-level.

PV_k^i is the path view for the border vertices in G_k^i. The path view for the border vertices contains the information required to retrieve a shortest path only between any pair of border vertices in a k-level region i. The function $PV_k^i(u,v)$ returns the predecessor of vertex v in the shortest path from vertex u. L_k^i is the set of edges linking all pairs of border vertices that are connected by a path in the subgraph G_k^i and L_k denotes these sets for the k-level. Each edge $(u,v)_i$ in L_k^i represents a shortest path from u to v in G_k^i. If there is another edge $(u,v)_j$ in L_k, then we just retain the edge with the minimum cost. The relation T in a particular k-level network represents the topological relation between k-level regions such that $(i,j) \in T$ if and only if the intersection set of vertices $B_k^{i,j} = B_k^i \cap B_k^j$ is not empty, where i and j are indices for different k-level regions.

Lemma 1. *Since the number of vertices embedded in a k-level region corresponding to subgraph G_k^i is $O(c^k m)$, the number of border vertices $|B_k|$ is $O\left(\sqrt{c^k m}\right)$, where m is the number of vertices in a 0-level subgraph [5, 7, 11].*

Lemma 2. *If $k > 0$, the number of vertices $|V_k|$ in a k-level subgraph G_k^i is $O\left(\sqrt{c^{k+1} m}\right)$ and the number of edges $|E_k|$ is $O\left(c^k m\right)$.*

A hierarchical shortest path algorithm creates shortest path tree layers PT_k^+ (for promotion) and PT_k^- (for demotion) at each k-level of the hierarchy $\langle G_0, G_1, \ldots, G_h \rangle$ of networks, where G_k represents a level $\langle G_k^0, G_k^1, \ldots, G_k^{r_k - 1} \rangle$ in the

[3] We assume without loss of generality that r is a power of c, *i.e.*, $r = c^h$, where h is the highest level in the hierarchy of networks.

hierarchy. We define a *shortest path tree* as a tree whose unique simple path from root to any vertex represents a shortest path. A *layer* of a shortest path tree is a subset of a shortest path tree contained in a particular level of the hierarchy of networks (see Fig. 1). The shortest path tree layers are computed in order to find the *hierarchical shortest path* from a source vertex s to a destination vertex d throughout all levels of the hierarchy. The algorithm uses a set S that represents source vertices for a layer at a k-level, where each vertex in S is associated with a cost.

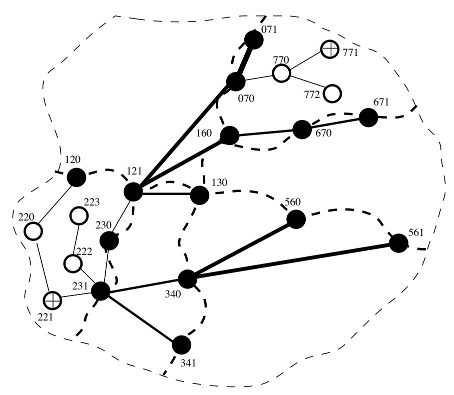


Fig. 1. Shortest path tree layers in a network subdivided into six regions. The source and destination vertices are depicted with a cross inside a circle. Border vertices are shown as filled black circles. Only the regions containing the source and destination vertices are depicted with all vertices. The other regions are shown only through border vertices.

3.1 Expanding Subpaths in Higher Levels

A *hierarchical shortest path* from s to d consists of a sequence of subpaths $\langle P_0^+(s, v_1), P_1^+(v_1, v_2), P_2^+(v_2, v_3), \ldots, P_{h-1}^+(v_{h-1}, v_h), P_h^-(v_h, v_{h+1}),$ $P_{h-1}^-(v_{h+1}, v_{h+2}), \ldots, P_2^-(v_{2h-2}, v_{2h-1}), P_1^-(v_{2h-1}, v_{2h}), P_0^-(v_{2h}, d)\rangle$, where a subpath from vertex v_i to vertex v_j in the k-level network is denoted by either $P_k^+(v_i, v_j)$ or $P_k^-(v_i, v_j)$. In order to have a complete shortest path, a hierarchical algorithm executes intra-regional queries for each k-level expanding subpaths at the k-level to subpaths at the next lower level until the whole path consists only of edges in the lowest level network.

The `Expand-Path` algorithm given below finds a whole shortest path from s to d represented by a sequence of edges in the lowest level network. The algorithm traverses the shortest path tree layers $PT_k^\pm$ in order to retrieve the subpaths that compose a shortest path from s to d at all levels of the hierarchy. Then, the algorithm performs intra-regional queries to expand each edge of these subpaths into a path P at a lower level.

Algorithm `Expand-Path`$(\langle PT_0^+, \ldots, PT_{h-1}^+, PT_h^-, PT_{h-1}^-, \ldots, PT_0^- \rangle, s, d)$

```
1. $P_0^-(v_{2h}, d) \leftarrow$ Traverse-Backwards$\big(PT_0^-, d\big)$
2. for $k \leftarrow 1$ to $h$ do
   (a) $P_k^-(v_{2h-k}, v_{2h-k+1}) \leftarrow$ Traverse-Backwards$\big(PT_k^-, v_{2h-k+1}\big)$
3. for $k \leftarrow h-1$ to $1$ do
   (a) $P_k^+(v_k, v_{k+1}) \leftarrow$ Traverse-Backwards$\big(PT_k^+, v_{k+1}\big)$
4. $P_0^+(s, v_1) \leftarrow$ Traverse-Backwards$\big(PT_0^+, v_1\big)$
5. for $k \leftarrow h$ to $1$ do
   (a) $P \leftarrow \emptyset$
   (b) for $u \leftarrow v_k \wedge (u,v)_i \in P_k^-(v_k, v_{2h-k+1})$ to $v_{2h-k+1}$ do
       i. $P \leftarrow P \oplus$ Intra-Regional$(u, v, k, i)$
   (c) $P_{k-1}^-(v_{k-1}, v_{2h-k+2}) \leftarrow P_{k-1}^+(v_{k-1}, v_k) \oplus P \oplus P_{k-1}^-(v_{2h-k+1}, v_{2h-k+2})$
```

Initially, the algorithm traverses each shortest path tree layer backwards using the procedure `Traverse-Backwards` (steps 1, 2, 3, and 4). This procedure creates the subpaths $P_k^+(v_j, v_{j+1})$ and $P_k^-(v_j, v_{j+1})$ at each k-level by retrieving a sequence of edges $(u,v)_i$ for each subpath. The algorithm expands each subpath starting at the highest level until it finds a whole path represented by edges only at the lowest level (step 5). For each k-level, the algorithm takes the corresponding subpath $P_k^-(v_k, v_{2h-k+1})$ and the procedure `Intra-Regional` expands each edge $(u,v)_i$ in this subpath into a subpath in the $(k-1)$-level (step 5.b). This procedure finds the subpath from u to v in the subgraph G_{k-1}^i corresponding to a region i at the $(k-1)$-level. These subpaths at the $(k-1)$-level are concatenated (operator $\oplus$) into a path P (step 5.b.i). Then, all subpaths at the $(k-1)$-level are concatenated into only one subpath $P_{k-1}^-(v_{k-1}, v_{2h-k+2})$ (step 5.c).

An important issue in the complexity analysis of the `Expand-Path` algorithm is the number of edges that an expanded shortest path will have at a particular level of the hierarchy.

Lemma 3. *The number of edges in a shortest path expanded to the $(h-i)$-level is $O(2^i)$.*

For intra-regional queries, we may use any flat strategy. However, we propose an efficient method for this task in the next section. A path view strategy precomputes all shortest paths in the network for each region. When $c = 2$, this strategy requires $O(m)$ time to retrieve a shortest path at the lowest level and $O(1)$ time otherwise. The path view strategy requires $O(m^{1.5})$ space at the lowest level and $O(|B_k|^2) = O(c^k m)$ otherwise. The path view at the highest level and the expanded shortest path at the lowest level require $O(n)$ space. They dominate the space requirement for `Expand-Path`.

Theorem 1. *If the intra-regional query is implemented as a path view approach, the algorithm* `Expand-Path` *requires $O(n)$ time.*

The path view in the highest level and the expanded shortest path in the lowest level require $O(n)$ space. They dominate the space requirement for the algorithm `Expand-Path`.

Theorem 2. *If the intra-regional query is implemented as a path view approach, the algorithm* `Expand-Path` *requires* $O(n + m^{1.5})$ *space.*

Another flat strategy for intra-regional queries is a single-source shortest path algorithm [4]. This strategy needs $O(|V_k| \log |V_k| + |E_k|)$ time to find a shortest path in a k-level subgraph and requires $O(|V_k| + |E_k|)$ space.

Theorem 3. *If the intra-regional query is implemented as a single-source shortest path algorithm, the algorithm* `Expand-Path` *requires* $O(n \log m)$ *time.*

The single-source shortest path algorithm requires space for two subgraphs at each level of the hierarchy (except the highest) and for the expanded path P. The total space required is $O(n)$. Therefore, there is no improvement concerning space requirement when a single-source shortest path algorithm performs intra-regional queries in a hierarchical approach.

Theorem 4. *If the intra-regional query is implemented as a single-source shortest path algorithm, the algorithm* `Expand-Path` *requires* $O(n)$ *space.*

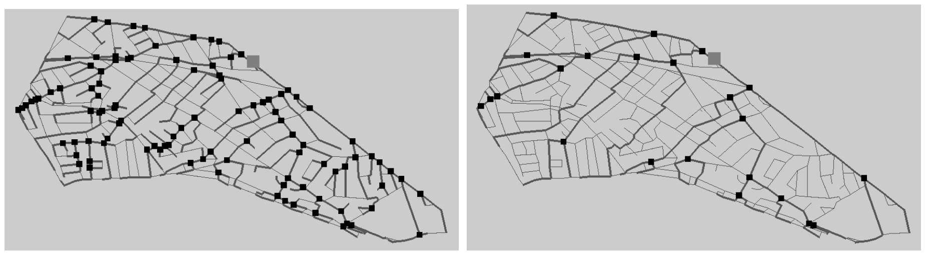
4 The Hybrid Shortest Path Algorithm

Given an edge $e = (u, v)_i$ at the k-level network ($k > 0$), an intra-regional query consists of expanding the edge e into a subpath from u to v in the subgraph G^i_{k-1} at the $(k-1)$-level, where u and v are border vertices in G^i_{k-1}. The query may be performed using any flat strategy, ranging from a single-source shortest path algorithm to a lookup in a path view for border vertices.

The strategy proposed in this paper uses a hybrid path view for border vertices. The hybrid path view represents each shortest path between border vertices of a region by just a sequence of guide vertices. A usual path view is implemented as a predecessor matrix, where each row of the predecessor matrix represents a shortest path tree corresponding to a particular source vertex as the root. Each entry of the matrix row specifies the predecessor vertex for a vertex in the shortest path tree. A *guide vertex* is a vertex acting as predecessor vertex for more than one vertex in the predecessor matrix implementing the path view for border vertices. The path view for border vertices consists of shortest path trees composed only of shortest paths from a border vertex to the other border vertices (see Fig. 2).

Lemma 4. *The number of guide vertices in a shortest path tree of a path view for a subgraph at the k-level is* $O(\sqrt{c^k m})$.

The hybrid path view $\overline{PV^i_k}$ for each region i at each k-level of the network hierarchy is retrieved from the corresponding path view PV^i_k for border vertices. In order to create $\overline{PV^i_k}$, each guide vertex is identified in a traversal of



(a) Destinations are all vertices. (b) Destinations are only border vertices.

Fig. 2. Shortest path tree from a source vertex (gray square) and guide vertices (black squares) in a lowest level region.

PV_k^i. A hybrid path view is implemented as a predecessor sparse matrix whose columns only consider guide vertices $\overline{V_k^i}$ and border vertices B_k^i. Furthermore, the predecessor relation in this sparse matrix is only expressed in terms of guide vertices.

The algorithm `Hybrid-View` below creates a hybrid path view $\overline{PV_k^i}$ from the path view PV_k^i. Each path view PV_k^i is associated with a set of vertices V_k^i and a set of border vertices B_k^i. The algorithm builds a matrix Γ to keep track of the shortest path tree considering only paths to border vertices. The algorithm uses a vector Λ to store the number of times any vertex $v \in V_k^i$ is a predecessor in a particular shortest path tree of PV_k^i.

Algorithm `Hybrid-View`(PV_k^i)

```
1. for $u \in B_k^i$ do
   (a) $\overline{V_k^i} \leftarrow B_k^i$
   (b) $\Lambda \leftarrow 0$
   (c) $\Gamma \leftarrow$ false
   (d) for $v \in B_k^i$ do
       i. $\Gamma(u,v) \leftarrow$ true
       ii. $v' \leftarrow PV_k^i(u,v)$
       iii. while $v' \neq u$ do
            A. $\Gamma(u,v') \leftarrow$ true
            B. $v' \leftarrow PV_k^i(u,v')$
   (e) for $v \in V_k^i$ do
       i. if $\Gamma(u,v)$ then
          A. $\Lambda(PV_k^i(u,v)) \leftarrow \Lambda(PV_k^i(u,v)) + 1$
   (f) for $v \in V_k^i$ do
       i. if $\Lambda(v) > 1$ then
          A. $\overline{V_k^i} \leftarrow \overline{V_k^i} \cup v$
   (g) for $v \in \overline{V_k^i}$ do
       i. $v' \leftarrow PV_k^i(u,v)$
```

```
ii. while $v' \notin \overline{V_k^i}$ do
    A. $v' \leftarrow PV_k^i(u, v')$
iii. $\overline{PV_k^i}(u, v) \leftarrow v'$
```

`Hybrid-View` algorithm traverses the path view PV_k^i computing the hybrid path for each vertex $u \in B_k^i$ corresponding to a shortest path tree in PV_k^i. Initially, the algorithm finds the shortest path tree considering only paths to border vertices (step 1.d). The matrix Γ identifies the edges belonging to this shortest path tree (see Fig. 2.b). According to Γ, the algorithm computes the vector Λ that stores the number of sons for each vertex in the shortest path tree considering only paths to border vertices (step 1.e). Each vertex $v \in V_k^i$ that is a predecessor for more than one vertex in a shortest path tree is considered a guide vertex and inserted in $\overline{V_k^i}$ (step 1.f). Next, the algorithm computes the corresponding shortest path tree only in terms of guide vertices in $\overline{V_k^i}$ (step 1.g). The predecessor of a vertex in $\overline{V_k^i}$ is the first vertex in a backward traversal of the corresponding shortest path tree in PV_k^i that belongs to $\overline{V_k^i}$.

The hybrid path view computation implies pre-processing time. The time required for the `Hybrid-View` algorithm is dominated by the path view scanning that counts predecessors and finds guide vertices.

Theorem 5. *Algorithm* `Hybrid-View` *needs $O(m^2)$ time at the lowest level and $O((c^k m)^{1.5})$ time otherwise.*

The hybrid path view computation at all levels of the hierarchy requires additional pre-processing time $O(rm^2 + \sum_{k=1}^{h} \frac{r}{c^k}(c^k m)^{1.5})$. This expression is equivalent to $O(nm + n\sqrt{n}) = O(n(m + \sqrt{n}))$. If we assume $r = m = \sqrt{n}$, then the additional pre-processing time becomes $O(n^{1.5})$.

At higher levels, a hybrid path view has the same space requirements as a path view when $c = 2$. However, at the lowest level, a hybrid path view requires $O(m)$ space. This is an improvement compared to the $O(m^{1.5})$ space required by a path view at this level. This way, the `Hybrid-View` algorithm space requirements are dominated by the path view predecessor matrix.

Theorem 6. *Algorithm* `Hybrid-View` *requires $O(m^{1.5})$ space at the lowest level and $O(c^k m)$ space otherwise.*

An intra-regional query for an edge $(u, v)_i$ in a 1-level network is performed by the hybrid shortest path algorithm. This algorithm uses the subgraph G_0^i and the hybrid path view $\overline{PV_0^i}$ information in order to find a single shortest path P from u to v in the 0-level region i. $\overline{PV_0^i}$ describes the shortest path in terms of guide vertices as a partial shortest path $\overline{P_0^i}(u, v)$. Therefore, the algorithm expands $\overline{P_0^i}(u, v)$ finding a sequence of subpaths between consecutive guide vertices in $\overline{P_0^i}(u, v)$.

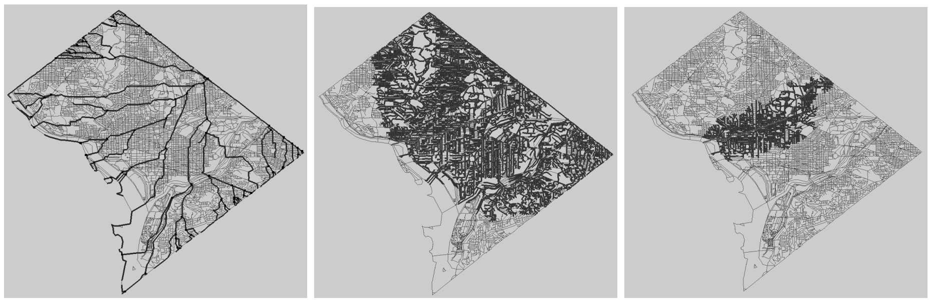
The algorithm `Hybrid-Shortest-Path` below creates a shortest path tree PT whose root represents the source vertex u. A priority queue Q is used to keep track of all information related to the current vertices in $V^i_{k-1} - PT$. Each entry $(u, f(u), e)$ in Q represents a vertex u with an estimate cost $f(u)$ and a predecessor edge e. The set Q' keeps track of the vertices in Q whose cost is not infinity and, consequently, will be reset for the next subpath search.

Algorithm `Hybrid-Shortest-Path`(u, v, k, i)

```
1. $\overline{P^i_0}(u,v) \leftarrow$ Path-Lookup$\left(\overline{PV^i_0}, u, v\right)$
2. $PT \leftarrow Q \leftarrow \emptyset$
3. $Q \leftarrow Q \cup (u, 0, 0)$
4. for $v' \in V^i_0 \wedge v' \neq u$ do
   (a) $Q \leftarrow Q \cup (v', \infty, \infty)$
5. for $(s,d) \leftarrow (u,v') \wedge (s,d) \in \overline{P^i_0}(u,v)$ to $(v'', v)$ do
   (a) $(u', f(u'), e') \leftarrow$Extract-Min$(Q)$
   (b) $Q' \leftarrow Q' - u'$
   (c) $PT \leftarrow PT \cup (u', f(u'), e')$
   (d) while $u' \neq d$ do
       i. for $(u', u'')_j \in E^i_0$
          A. if $f(u') + |(u', u'')_j| < f(u'')$ then
             – $f(u'') \leftarrow f(u') + |(u', u'')_j|$
             – $e'' \leftarrow (u', u'')_j$
             – $Q' \leftarrow Q' \cup u''$
       ii. $(u', f(u'), e') \leftarrow$Extract-Min$(Q)$
       iii. $Q' \leftarrow Q' - u'$
       iv. $PT \leftarrow PT \cup (u', f(u'), e')$
   (e) for $u' \in Q'$ do
       i. $f(u') \leftarrow e' \leftarrow \infty$
   (f) for $(d, u'')_j \in E^i_0$
       i. $f(u'') \leftarrow |(d, u'')_j|$
       ii. $e'' \leftarrow (d, u'')_j$
6. $P \leftarrow$Traverse-Backwards$(PT, v)$
```

First, the algorithm finds the partial shortest path $\overline{P^i_0}(u,v)$ using the procedure `Path-Lookup` which traverses the hybrid path view $\overline{PV^i_0}$ (step 1). Initially, PT is empty and Q has all vertices with cost and predecessor edge equal to ∞, but the source vertex u whose cost is 0 (steps 2, 3, and 4). Next, the algorithm finds a shortest subpath in G^i_0 corresponding to each edge (s,d) in the partial shortest path (step 5). The shortest subpath for each edge is computed in a similar way to Dijkstra‘s shortest path algorithm. However, the current state of the priority queue Q means that there is no need to consider the vertices already in PT for the current subpath. Therefore, each subsequent search has an initial Q just with all remaining vertices (step 5.e). All costs and predecessor edges are set to ∞, but vertices connected to d have its costs and predecessor edges updated to represent that the next source vertex is d with cost equal to 0 (step 5.f). The subpath from u to v is computed by traversing PT from v using the procedure `Traverse-Backwards` (step 6).

In the worst case, the hybrid shortest path algorithm has the same time complexity as Dijkstra's single-source shortest path algorithm [4]. However, the worst case only happens when all vertices of the subgraph are visited by the algorithm. The number of vertices visited by the hybrid shortest path algorithm is much smaller than the number of vertices of the subgraph when vertices are uniformly distributed in the region corresponding to the subgraph and the guide vertices are uniformly distributed along a shortest path (see Fig. 3).



(a) Shortest path tree for border vertices. (b) Dijkstra's algorithm. (c) Our hybrid algorithm.

Fig. 3. Vertices visited by shortest path algorithms.

Theorem 7. *Assume a uniform distribution of vertices at the lowest level and the guide vertices are also uniformly distributed along a shortest path. The number of vertices visited by the hybrid shortest path algorithm is* $O(\sqrt{m})$.

Assuming that vertices are uniformly distributed, the running time for the hybrid shortest path algorithm becomes $O(m)$ at the lowest level.

Theorem 8. *Assume a uniform distribution of vertices at the lowest level and the guide vertices are also uniformly distributed along a shortest path. The hybrid shortest path algorithm spends* $O(m)$ *time at the lowest level.*

Since the subgraph G_0^i and the hybrid path view $\overline{PV_0^i}$ are the inputs for the hybrid shortest path algorithm, the space requirements for the algorithm are the same as for a single-source shortest path algorithm.

Theorem 9. *The hybrid shortest path algorithm requires* $O(m)$ *space in the* k*-level when* $k = 1$ *and* $O(c^{k-1}m)$ *space otherwise.*

We have now seen that the best time and space requirements for intra-regional queries at the lowest level are achieved by the hybrid shortest path algorithm. On the other hand, for higher levels, the best strategy concerning time and space requirements is the path view approach when $c = 2$. Therefore, the expansion of the hierarchical path into a path with edges at the lowest level requires $O(n)$ time and space.

5 Experimental Results

We evaluate the time and space performance of our hybrid approach using road networks from the TIGER files of the U.S. Census Bureau. We compare our method with two state-of-the-art techniques for shortest path finding on spatial networks: the shortest path quadtrees [13] and a flat approach based on Dijkstra's algorithm [4]. All algorithms were implemented using C++ in a Dell XPS 730X machine with a i7 CPU at 3.20GHz and 6Gb of RAM. All data structures necessary for the execution of all algorithms were loaded in the main memory.

The implementation of our approach consists of four pre-processing modules: network, border, view, and hierarchy. They pre-process the TIGER/Line files in order to find the information required by the shortest path algorithms. A road network for each county in a state is obtained in the network module. After that, the border module finds for each county the set of border vertices that are shared with another county. Then, the view module computes the path view for each county. Finally, the hierarchy module finds a higher-level network for the state to be used in a hierarchical shortest path algorithm.

In our experimental setup, we initially selected a spatial network that requires only the amount of available space for all evaluated techniques. To obtain networks of smaller sizes, we determined the centroid of the original network and incrementally removed vertices according to their distance to the centroid. This way, we were able to generate networks of continuous sizes to evaluate the space requirements of all considered methods for a continuous range of network sizes. Similarly, for each network size, we compute shortest paths from the centroid vertex to the most distant vertex in the spatial network. This way, we investigated the time needed to found shortest paths of varying lengths. The memory space and the running time obtained in our experiments is shown in Fig. 4. The space requirements for our hybrid approach a virtually the same as the requirements of the Dijkstra's algorithm. The shortest path quadtree spends more space as expected from its space complexity $O(n^{1.5})$. In terms of running time, the hybrid approach performs best followed closely by the shortest path quadtree method.

6 Conclusion

We introduced a hybrid shortest path algorithm to perform intra-regional queries. This strategy uses a subsequence of pre-processed vertices that belong to the shortest path while actually computing the whole shortest path. At the lowest level, the hybrid algorithm requires $O(m)$ time and space assuming a uniform

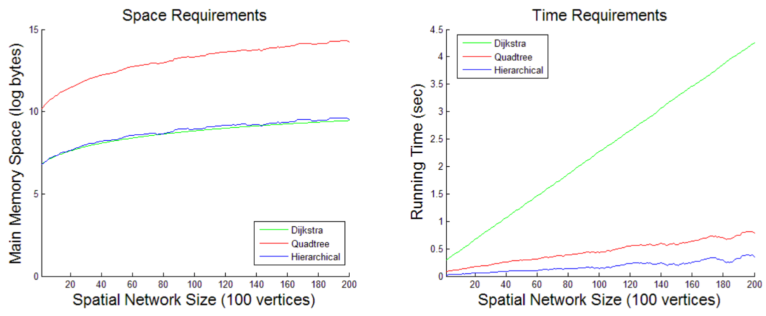


Fig. 4. The memory space and running time of our hybrid approach compared to other path finding methods.

distribution of vertices. For higher levels, the path view approach takes $O(1)$ time and requires $O(c^k m)$ space.

References

1. Aref, W.G., Samet, H.: Efficient processing of window queries in the pyramid data structure. In: Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS). pp. 265–272. Nashville, TN (Apr 1990)
2. Chen, Z., Shen, H., Zhou, X., Yu, J.: Monitoring path nearest neighbor in road networks. In: SIGMOD'09, Proceedings of the 35th SIGMOD International Conference on Management of Data. pp. 591–602 (2009)
3. Demiryurek, U., Banaei-Kashani, F., Shahabi, C.: Efficient continuous nearest neighbor query in spatial networks using Euclidean restriction. In: SSTD'09, Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases. Lecture Notes in Computer Science, vol. 5644, pp. 25–43. Springer-Verlag (2009)
4. Dijkstra, E.: A note on two problems in connection with graphs. Numerische Mathematik 1, 269–271 (1959)
5. Frederickson, G.: Fast algorithms for shortest paths in planar graphs, with applications. SIAM J. Comput. 16(6), 1004–1022 (December 1987)
6. Goldberg, A., Harrelson, C.: Computing the shortest path: a^* search meets graph theory. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 156–165 (Jan 2005)
7. Goodrich, M.: Planar separators and parallel polygon triangulation. In: Proceedings of the 24th ACM Symposium on Theory of Computing. pp. 507–516. ACM (May 1992)
8. Henzinger, M., Klein, P., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. Journal of Computer and System Sciences 55(1), 3–23 (August 1997)
9. Jing, N., Huang, Y.W., Rundensteiner, E.: Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. IEEE Transactions on Knowledge and Data Engineering 10(3), 409–432 (1998)

10. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: Proceedings of the 30th International Conference on Very Large Databases. pp. 840–851 (2004)
11. Lipton, R., Tarjan, R.: A separator theorem for planar graphs. SIAM J. Appl. Math 36, 177–189 (1979)
12. Samet, H., Alborzi, H., Brabec, F., Esperança, C., Hjaltason, G.R., Morgan, F., Tanin, E.: Use of the SAND spatial browser for digital government applications. Commun. ACM 46(1), 63–66 (Jan 2003)
13. Samet, H., Sankaranarayanan, J., Alborzi, H.: Scalable network distance browsing in spatial databases. In: SIGMOD'08, Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data. pp. 43–54 (2011)
14. Sankaranarayanan, J., Alborzi, H., Samet, H.: Efficient query processing on spatial networks. In: Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems. pp. 200–209. Bremen, Germany (Nov 2005)
15. Sankaranarayanan, J., Alborzi, H., Samet, H.: Distance join queries on spatial networks. In: Proceedings of the 14th ACM International Symposium on Advances in Geographic Information Systems. pp. 211–218. Arlington, VA (Nov 2006)
16. Sankaranarayanan, J., Samet, H.: Distance oracles for spatial networks. pp. 652–663. Shanghai, China (Apr 2009)
17. Sankaranarayanan, J., Samet, H.: Query processing using distance oracles for spatial networks 22(8), 1158–1175 (Aug 2010), best Papers of ICDE 2009 Special Issue
18. Sankaranarayanan, J., Samet, H.: Roads belong in databases 33(2), 4–11 (Jun 2010), invited paper.
19. Sankaranarayanan, J., Samet, H., Alborzi, H.: Path oracles for spatial networks. In: Proceedings of the VLDB Endowment. pp. 1210–1221 (2009)
20. Shaffer, C.A., Samet, H., Nelson, R.C.: QUILT: a geographic information system based on quadtrees 4(2), 103–131 (April–June 1990), also University of Maryland Computer Science Technical Report TR–1885.1, July 1987
21. Shekhar, S., Fetterer, A., Goyal, B.: Materialization trade-offs in hierarchical shortest path algorithms. In: Scholl, M., Voisard, A. (eds.) SSD'97, Proceedings of the 5th International Symposium on Advances in Spatial Databases. Lecture Notes in Computer Science, vol. 1262, pp. 94–111. Springer verlag (1997)
22. Sivan, R., Samet, H.: Algorithms for constructing quadtree surface maps. In: Proceedings of the 5th International Symposium on Spatial Data Handling. vol. 1, pp. 361–370. Charleston, SC (Aug 1992)