A Linear Iterative Approach for Hierarchical Shortest Path Finding *

Gutemberg Guerra Filho
Department of Computer Science
University of Maryland, College Park 20742
guerra@cs.umd.edu

Hanan Samet
Department of Computer Science
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland, College Park 20742
hjs@umiacs.umd.edu

ABSTRACT

We present a hierarchical approach that subdivides a network with n vertices into r regions with the same number m of vertices ($n = rm$) and creates higher levels merging a constant number c of adjacent regions. We propose linear iterative algorithms to find a shortest path and to expand this path into the lowest level. Since our approach is non-recursive, the complexity constants are small and the algorithms are more efficient in practice than other recursive optimal approaches. A hybrid shortest path algorithm to perform intra-regional queries in the lowest level is introduced. This strategy uses a subsequence of vertices that belong to the shortest path while actually computing the whole shortest path. The hybrid algorithm requires $O(m)$ time and space assuming an uniform distribution of vertices. This represents a further improvement concerning space, since a path view approach requires $O(m^{1.5})$ space in the lowest level. For higher k-levels, a path view approach spends $O(1)$ 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 centralized path finding architecture for such information systems consists in a processing center that receives path queries in a road network and responds in real-time. The real-time performance is a requirement for the system effectiveness since a driver may take a next turn within a few seconds. A large number of queries in a huge network may prevent the system from meeting the real-time requirement when a flat path finding approach is used.

A *flat approach* does not subdivide the original graph that represents the network. This approach solves the path finding problem either by executing a single-source shortest path algorithm in the whole network [2, 3] or retrieving a path with one lookup in a pre-computed path view. Dijkstra's shortest path algorithm requires $O(n \log n)$ time and $O(n)$ space in planar graphs [7, 10].

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 represents the path. A path query requires $O(n^2)$ space and spends $O(n)$ time, 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, 11].

Large networks may need an unacceptable amount of time and space in order to satisfy the real-time constraint. Therefore, a hierarchical approach is applied. A *hierarchical approach* subdivides a single network into smaller regions. This process creates a hierarchy of multilevel networks for a path finding search. We consider the single-pair shortest path problem for planar graphs with non-negative edge weights. The problem consists in finding a shortest path from a given source vertex s to a given destination vertex d in a planar graph.

In this paper, we present a hierarchical approach that subdivides the network into r 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 in the other $O(\log r)$ levels by creating a higher level merging c adjacent regions. Therefore, the parameters $\langle r, m, c\rangle$ completely define the structure of a simple hierarchy of networks.

We introduce iterative (non-recursive) algorithms to find a shortest path through all levels of the $\langle r, m, c\rangle$ hierarchy and to expand this shortest path into a shortest path composed by edges only in the lowest level that represents the original network.

Given a hierarchy of networks, the proposed hierarchical shortest path algorithm starts in the lowest level network

*This research was supported in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES.

STOC '03 San Diego, California USA

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 until it reaches the highest level. Then, the search is demoted to the next lower level towards the destination vertex. The demotion step is performed until it reaches the lowest level network. A resulting *hierarchical shortest path* consists of subpaths in each level of the network. The hierarchical approach executes *intra-regional queries* for each level in order to expand these subpaths to the next lower level. The queries are performed until the whole path is represented by subpaths in the lowest level network.

In the worst case, our hierarchical approach requires $O(n)$ query time and space, where $n = rm$ is the total number of vertices in the original network. These time upper bounds are achieved using a hierarchy definition where $r = m = \sqrt{n}$ and $c = 2$. If we assume that pre-processing to build the hierarchy of networks is performed a neglected number of times concerning the number of queries, than our approach would require optimal time and space resources. Further, since our approach is non-recursive, the complexity constants are small, what implicates more efficiency in practice when compared to other recursive optimal approaches.

We also propose a new strategy to execute intra-regional queries in the hierarchical approach. This strategy is based on an hybrid shortest path algorithm that uses a partial path in order to find a single shortest path between two vertices on the border of a region. A partial path is a subsequence of vertices that belong to the shortest path.

Using the hybrid shortest path algorithm, the time and space requirements for intra-regional queries in the lowest level is $O(m)$. This is an improvement compared to the $O(m^{1.5})$ space required by a path view. For higher levels, we use a path view approach. Defining $c = 2$, the path view approach spends $O(1)$ time and requires $O(c^k m)$ space in a k-level network for $k > 0$.

This paper is organized as follows. In Section 2, we review work related to hierarchical path finding. The $\langle r, m, c\rangle$ hierarchical approach is presented in Section 3. We present the hybrid shortest path algorithm in Section 4. Section 5 discusses the main contributions of this paper.

2. RELATED WORK

Frederickson [5] proposed a hierarchical algorithm for the single-source shortest path problem on planar graphs. The algorithm takes $O(n\sqrt{\log n})$ running time, where n is the number of vertices in a planar graph. The algorithm is based on Dijkstra's algorithm and uses a division of the planar graph into a three-level hierarchy of regions. A suitable graph division is created in $O(n \log n)$ pre-processing time using a technique based on a planar separator algorithm [12]. The algorithm depends on the fact that planar graphs have separators with size $O(\sqrt{n})$. 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. The all-pairs shortest path problem is solved in $O(n^2)$ time and space.

Frederickson [6] developed an algorithm for computing all-pairs shortest paths in a directed planar graph with real-valued edge cost and no negative cycles. The algorithm takes $O(pn)$ time, where p is the minimum cardinality of a subset of the faces that cover all vertices, taken over all planar embeddings of the planar graph.

Klein *et al.* [11] presented an optimal single-source shortest path algorithm for undirected planar graphs with non-negative edge weights that requires $O(n)$ time and space. Their recursive approach leads to big constants what in practice represents some additional time cost. The algorithm also uses separators, but they are not required to have size $O(\sqrt{n})$. They divide the graph into regions in $O(\log^* n)$ levels such that the lowest level regions consist of a single edge. The efficiency of the algorithm is due to the fact that relax operations are performed using smaller priority queues. The algorithm chooses a region, perform a number of relax operations depending on the level of the region, and abandons that region. A relax operation may use a node whose label is not the correct shortest path distance to that node. Therefore, a region may be selected again. They also compute the all-pairs shortest paths in $O(n^2)$ time and space.

Djidjev [4] solves the single-pair shortest path problem using a subdivision whose lowest level regions contain a single vertex. This approach requires $O(n^{1.5})$ pre-processing time and space. Djidjev presents an algorithm to compute only the shortest path distance between s and d in $O(\sqrt{n})$ time. The algorithm locates the vertices s and d in lowest level regions, find a nearest common ancestor (NCA) region and computes the minimum distance considering the vertices in the NCA region separator.

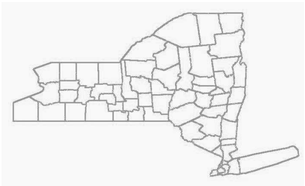
Jing *et al.* [9] suggested a path view approach that stores the direct successor vertex and the cost of a shortest path for each source-destination pair in a region. Therefore, the path view requires $O(m^2)$ space for a region in the lowest level, where m is the number of vertices in a single region. They use a path finding algorithm that queries recursively the shortest path cost through all levels in the hierarchy of networks. First, it determines 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 form 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 in the lowest level.

Shekhar *et al.* [13] 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 complexity analysis concerning the space required for the path views. The space storage required for the path views is $O(n^{5/3})$.

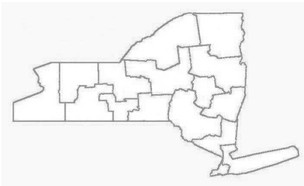
Chen and Xu [1] solves the single-pair shortest path problem in undirected planar graphs with non-negative edge weights. They take advantage of the planarity of general planar graphs to compute the minimum length among paths passing through pairs of border vertices as in the path finding algorithm used by Jing *et al.* [9]. They use a divide-and-conquer scheme to search a pair of border vertices more efficiently. A single shortest path cost is computed in time $O((|B_s| + |B_d|)\log(|B_s| + |B_d|))$ using a *frame search.*

3. $\langle R, M, C\rangle$ HIERARCHICAL APPROACH

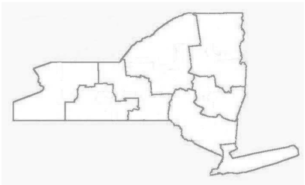
A *hierarchical approach* is based on the subdivision of the original network into regions (see Fig. 1). A region corresponds to a connected subgraph of the graph representing the network. A higher level network consists only of border vertices. A *border vertex* is a vertex that belongs to at least 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.



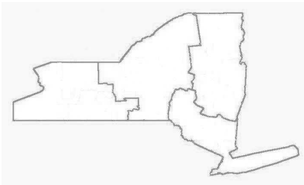
(a) 0-level network regions.



(b) 1-level network regions.



(c) 2-level network regions.



(d) 3-level network regions.

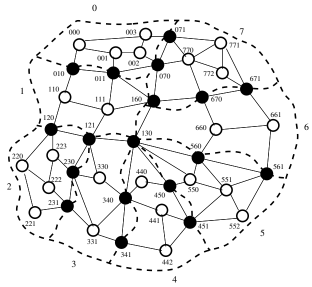
Figure 1: A hierarchy of network regions with four levels.

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.

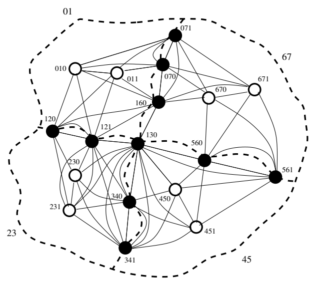
Each 0-level subgraph has m vertices and form a suitable subdivision of the graph G where boundaries of each corresponding region has a size of $O(\sqrt{m})$ vertices. Such suitable subdivision is obtained in $O(n \log n)$ time using a fragmentation algorithm [5]. Goodrich [8] proposed an algorithm to find a separator decomposition and a suitable subdivision in $O(n)$ time. The subdivision into subgraphs decreases the size of path views, thereby reducing the space and maintenance costs. The best cost is achieved when the subgraphs have approximately the same number of vertices. Therefore, we enforce the generation of a higher level to be constrained in such a way that each region will have about the same number of vertices.

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 (see Fig. 2).

A hierarchy of networks for a network represented by a graph G 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[1]. This way, all regions in 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 hierarchical level 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.



(a) 0-level network.



(b) 1-level network.

Figure 2: Two levels of a hierarchy of networks. Each region is surrounded by dashed lines and the border vertices are represented by filled circles.

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 keep the edge with minimum cost.

In order to analyze performance issues for the hierarchical approach, a complexity measure for a set of border vertices in a k-level network is required. A class of graphs satisfies an $f(n)$ separator theorem, if there exists a set of $O(f(n))$ vertices whose removal divides a graph from the class into two parts with at most αn vertices each for some constant $1/2 \leq \alpha < 1$. Lipton and Tarjan [12] found a set of nodes of size $O(\sqrt{n})$ whose removal divides a planar graph into regions with size less than $\frac{2}{3}n$. Therefore, the class of planar graphs is $\sqrt{n}$-separable. An m-division of a planar graph with n vertices is a division into $O(n/m)$ regions, each containing at most m vertices including $O(\sqrt{m})$ border vertices [5]. Such a recursive separator decomposition can be found

[1]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.

in $O(n)$ time [8]. The number of border vertices in any k-level comes from the m-division of a planar graph.

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.*

Proof. See [5, 8, 12]. □

Once a set of border vertices in a k-level network has a complexity measure, we can estimate the size of a subgraph in a higher level. Actually, we obtain complexity measures for sets of vertices V_k and edges E_k in a k-level subgraph G_k^i. In order to prove these complexity bounds, we introduce the *subgraph adjacency* concept. Two subgraphs are adjacent to each other if the intersection between the respective set of vertices is not empty.

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 $O\left(c^k m\right)$.*

Proof. The vertices in G_k^i correspond to the border vertices of c subgraphs in the $(k-1)$-level. Hence, $|V_k|$ is $O\left(c|B_{k-1}|\right)$ according to Lemma 1, and $|V_k| = O\left(\sqrt{c^{k+1} m}\right)$. In the worst case, the subgraphs in the $(k-1)$-level are strongly connected and, consequently, the number of edges $|E_k|$ is $O\left(c|B_{k-1}|^2\right) = O\left(c^k m\right)$. □

The lowest level of the hierarchy is composed by planar subgraphs. Hence, a single-source shortest path algorithm requires $O(m \log m)$ time in the lowest level. A higher level $(k > 0)$ subgraph G_k^i in the k-level is not planar and consists of the union of c cliques corresponding to c regions in the $(k-1)$-level. Therefore, Dijkstra's shortest path algorithm performance is different in a higher level.

LEMMA 3. *Defining $c = 2$, Dijkstra's algorithm spends time $O(c^k m)$ in a k-level subgraph G_k^i when $k > 0$.*

Proof. Dijkstra's algorithm requires $O\left(|V_k| \log |V_k| + |E_k|\right)$ time. Since $\log |V_k| = O(\sqrt{|V_k|})$, then $|V_k| \log |V_k| = O\left(|V_k|\sqrt{|V_k|}\right)$. We prove that $|V_k|\sqrt{|V_k|} = O(|E_k|)$, since $c|B_{k-1}|\sqrt{c|B_{k-1}|} < c|B_{k-1}|^2$. This inequation is equivalent to $\sqrt{c|B_{k-1}|} < |B_{k-1}|$. Squaring and simplifying this expression, we have $c < |B_{k-1}|$. Lemma 1 states that $|B_{k-1}| = O(\sqrt{c^{k-1} m})$, consequently, we have $c = O(\sqrt{c^{k-1} m})$. $|B_{k-1}|$ has the least value when $k = 1$, then $c = O(\sqrt{m})$. Therefore, Dijkstra's algorithm requires $O(|E_k|) = O(c^k m)$ time in a higher level. □

3.1 Hierarchical Optimal Path Finding

In order to find a shortest path, our hierarchical approach starts from the source vertex in the 0-level network (see Fig. 3). Every time the search completely reaches the border of a k-level region, the search is promoted to the $(k+1)$-level starting from the vertices in this border. Since this procedure reaches the highest level, every time the search completely reaches the border of a $(k+1)$-level region containing the destination vertex, then the search is demoted to the k-level starting from these border vertices.

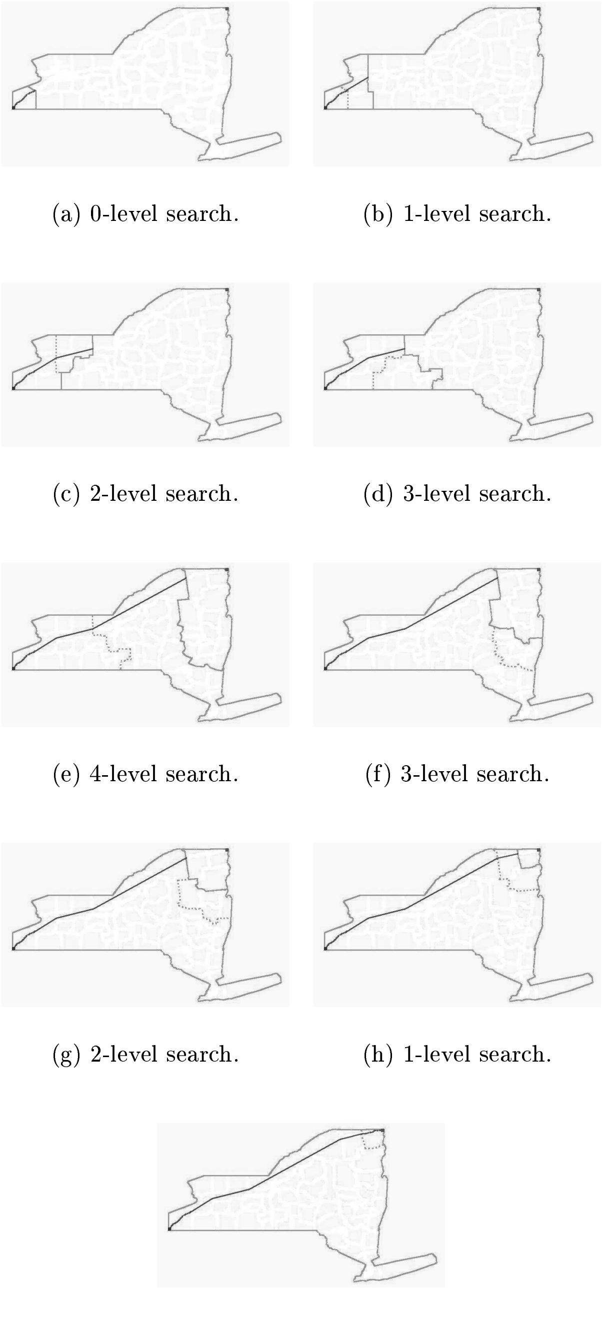
If the hierarchy of networks has $h+1$ levels, the search procedure executes a generalized shortest path algorithm $2h+1$ times in order to find the subpaths in each level of the hierarchy. The generalized algorithm is similar to Dijkstra's algorithm but allows initializing the priority queue that keeps unvisited vertices with more than one source vertex and initial costs greater than 0. The generalized shortest path algorithm computes the shortest path from a set S of source vertices to all vertices in a subgraph using only edges in this subgraph.

The hierarchical `Find-Path` algorithm below creates shortest path tree layers PT_k^+ (for promotion) and PT_k^- (for demotion) in 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 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. 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 in a k-level, where each vertex in S is associated with a cost.

Algorithm `Find-Path`$(\langle G_0, G_1, \ldots, G_h \rangle, s, d)$

1. $S \leftarrow (s, 0)$
2. **for** $k \leftarrow 0$ **to** $h-1$ **do**
   (a) $sr_k^+ \leftarrow$ `Locate-Region`(s, G_k)
   (b) $PT_k^+ \leftarrow$ `GSP`$(S, G_k^{sr_k^+})$
   (c) $S \leftarrow \emptyset$
   (d) **for** $u \in B_k^{sr_k^+}$ **do**
      i. $S \leftarrow S \cup (u,$ `Distance`$(PT_k^+, u))$
3. $PT_h^- \leftarrow$ `GSP`(S, G_h^0)
4. **for** $k \leftarrow h-1$ **to** 0 **do**
   (a) $sr_k^- \leftarrow$ `Locate-Region`(d, G_k)
   (b) $S \leftarrow \emptyset$
   (c) **for** $u \in B_k^{sr_k^-}$ **do**
      i. $S \leftarrow S \cup (u,$ `Distance`$(PT_{k+1}^-, u))$
   (d) $PT_k^- \leftarrow$ `GSP`$(S, G_k^{sr_k^-})$

Initially, the set S represents only the source vertex s with cost 0 (step 1). For each k-level but the highest one and starting in the lowest level (step 2), the algorithm locates s in a subgraph $G_k^{sr_k^+}$ using the `Locate-Region` procedure (step 2.a) and computes the shortest path tree layer PT_k^+ from S in this subgraph using the generalized single-source shortest path algorithm (`GSP`) (step 2.b). The next set S represents the set of border vertices $B_k^{sr_k^+}$ with the respective costs in PT_k^+ (steps 2.c and 2.d). The function `Distance` returns the distance from the source vertex s to the vertex u in the shortest path tree PT. Then, the shortest path tree layer



(a) 0-level search. (b) 1-level search.

(c) 2-level search. (d) 3-level search.

(e) 4-level search. (f) 3-level search.

(g) 2-level search. (h) 1-level search.

(i) 0-level search.

Figure 3: The shortest path search through a hierarchical network.

PT_h^- for the highest level is computed (step 3). Note that the highest level has just one shortest path tree layer. After that, for each k-level but the highest one and starting in the second highest level (step 4), the algorithm locates d in a subgraph $G_k^{sr_k^-}$ (step 4.a), the set S is updated to represent the border vertices $B_k^{sr_k^-}$ with the respective costs in PT_{k+1}^- (steps 4.b and 4.c). Finally, the shortest path tree layer PT_k^- from S in $G_k^{sr_k^-}$ is found (step 4.d).

THEOREM 1. *The algorithm* `Find-Path` *computes the shortest path distance from the source vertex s to the destination vertex d in G.*

Proof. Initially, the set S consists only of the source vertex s and the algorithm `GSP` computes the shortest path from s to all border vertices in the subgraph $G_0^{sr_0^+}$ that corresponds to the region containing s in the lowest level. Actually, the shortest paths are minimum only among those paths completely contained in $G_0^{sr_0^+}$. However, a shortest path from s to any border vertex in $G_0^{sr_0^+}$ may not be completely contained in a single lowest level region.

The shortest path from s to any border vertex u in $G_1^{sr_1^+}$ may include some other border vertices. Let v be the first border vertex from s to u in this shortest path. Hence, the subpath from v to u is a shortest subpath composed by subpaths whose source and destination vertices are border vertices. These subpaths are represented in $G_1^{sr_1^+}$ by edges with the same cost. Therefore, the shortest paths from s to all border vertices in $G_1^{sr_1^+}$ and completely contained in $G_1^{sr_1^+}$ is computed by the algorithm `GSP` by initializing S to the border vertices of $G_0^{sr_0^+}$ with the shortest path costs computed previously. The same step inductively shows that the algorithm `Find-Path` computes the shortest path cost from s to all border vertices in the highest level. Analogously, another induction provides that the algorithm computes the shortest path costs to border vertices towards the destination vertex d. Finally, a last step in the lowest level, computes the shortest path cost from s to d. $\square$

The generalized shortest path algorithm dominates the time requirements for the algorithm `Find-Path`.

THEOREM 2. *The algorithm* `Find-Path` *spends time* $O(m \log m + n)$.

Proof. The procedure `Locate-Region` finds a set of subgraphs in the k-level that may contain the source/destination vertex. Since these subgraphs represent connected regions embedded in a plane, this set has $O(1)$ subgraphs. Provided that the vertices of a subgraph are sorted, additional search takes $O(\log |V_k|)$ time to identify the region in the k-level where the source/destination vertex is located. Hence, the procedure `Locate-Region` spends $O(r_k + \log |V_k|)$ time. The procedure `GSP` spends time $O(|V_k| \log |V_k| + |E_k|)$, since basically it has the same time complexity as Dijkstra‘s algorithm. According to Lemma 3, the procedure `GSP` spends time $O(c^h m) = O(rm) = O(n)$ in the h-level. The loops in the algorithm execute the procedures `Locate-Region`, `GSP` and updates the source set S with the border vertices in

$B_k^{sr_k^{\pm}}$. Hence, each iteration spends time $O(r_k + \log |V_k| + |V_k| \log |V_k| + |E_k| + |B_k|)$. If $k = 0$, the loop iteration spends $O(r + m \log m)$ time, otherwise, it spends $O\left(r_k + c^k m\right)$ time. Therefore, the algorithm `Find-Path` spends total time

$$O\left((r + m \log m) + \sum_{k=1}^{h-1} \left(r_k + c^k m\right) + n\right).$$

This is equivalent to $O\left(m \log m + n + \frac{c(r-1)}{c-1} + \frac{(r-c)m}{c-1}\right)$. Therefore, the algorithm `Find-Path` spends $O(m \log m + n)$ time. □

The subgraph for the highest level requires $O(n)$ space. This is the input for the procedure `GSP` in the highest level and dominates the space requirement concerning the algorithm `Find-Path`.

THEOREM 3. *The algorithm* `Find-Path` *requires* $O(n)$ *space.*

Proof. The procedure `Locate-Region` uses the vertex sets V_k of $O(1)$ regions in the k-level. Hence, the procedure `Locate-Region` requires $O(|V_k|)$ space. This way, `Locate-Region` requires $O(m)$ space in the lowest level and $O(\sqrt{c^{k+1}m})$ space otherwise. The procedure `GSP` requires $O(|V_k| + |E_k|)$ space. Therefore, `GSP` requires $O(m)$ space in the lowest level and $O(c^k m)$ space otherwise. The priority queue S stores general source vertices that are only border vertices. Hence, S requires $O(|B_k|) = O(\sqrt{c^k m})$ space. Therefore, the algorithm requires $O(m)$ space for the lowest level, $O(c^k m)$ space for the k-level when $0 < k < h$, and $O(rm)$ space for the `GSP` procedure in the highest level. This way, the space requirement for the algorithm `Find-Path` is $O(rm) = O(n)$. □

If we define $r = m = \sqrt{n}$, the algorithm `Find-Path` spends $O(n)$ time. However, a shortest path from s to d is encoded in shortest path tree layers through all levels of the hierarchy and requires further processing to expand this shortest path into one composed by edges only in the lowest level (see Fig. 4).

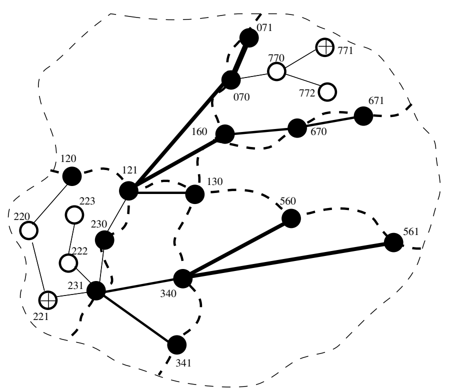
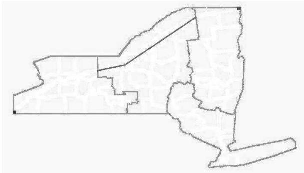


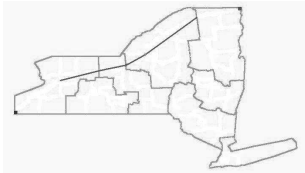
Figure 4: Shortest path tree layers.

3.2 Expanding Subpaths in Higher Levels

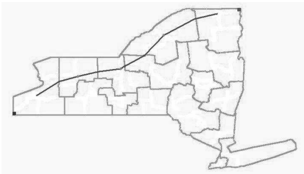
A shortest path from s to d consists of a sequence of subpaths $\langle P_0^+(s, v_1), P_1^+(v_1, v_2), P_2^+(v_2, v_3), \ldots, P_{h-1}^+(v_{h-1}, v_h),$ $P_h^-(v_h, v_{h+1}), P_{h-1}^-(v_{h+1}, v_{h+2}), \ldots, P_2^-(v_{2h-2}, v_{2h-1}),$ $P_1^-(v_{2h-1}, v_{2h}), P_0^-(v_{2h}, d)\rangle$, where a 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, the algorithm executes intra-regional queries for each k-level expanding subpaths in the k-level to subpaths in the next lower level until the whole path consists only of edges in the lowest level network (see Fig. 5).



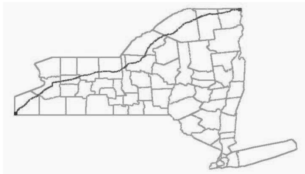
(a) 3-level shortest path.



(b) 2-level shortest path.



(c) 1-level shortest path.



(d) 0-level shortest path.

Figure 5: The expansion of a hierarchical shortest path through all levels of the hierarchy.

The `Expand-Path` algorithm 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 in all levels of the hierarchy. Then, the algorithm performs intra-regional queries to expand each edge of these subpaths into a path P in a lower level.

Algorithm `Expand-Path`$(PT_*^{\pm}, s, d)$

1. $P_0^-(v_{2h}, d) \leftarrow$ `TravBack`(PT_0^-, d)
2. **for** $k \leftarrow 1$ **to** h **do**
   (a) $P_k^-(v_{2h-k}, v_{2h-k+1}) \leftarrow$ `TravBack`(PT_k^-, v_{2h-k+1})
3. **for** $k \leftarrow h-1$ **to** 1 **do**
   (a) $P_k^+(v_k, v_{k+1}) \leftarrow$ `TravBack`(PT_k^+, v_{k+1})
4. $P_0^+(s, v_1) \leftarrow$ `TravBack`(PT_0^+, v_1)
5. **for** $k \leftarrow h$ **to** 1 **do**
   (a) $P \leftarrow \emptyset$
   (b) **for** each $(u, v)_i \in P_k^-(v_k, v_{2h-k+1})$ **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 `TravBack` (steps 1, 2, 3, and 4). The procedure creates the subpaths $P^+_k(v_j, v_{j+1})$ and $P^-_k(v_j, v_{j+1})$ in 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 in 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^i_{k-1} corresponding to a region i in the $(k-1)$-level. These subpaths in the $(k-1)$-level are concatenated (operator $\oplus$) into a path P (step 5.b.i). Then, all subpaths in 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 analysis of the `Expand-Path` algorithm is the number of edges that an expanded shortest path will have in a particular level of the hierarchy. Without loss of generality, we assume that the border vertices between two regions compose a shortest path. Hence, for higher levels, any shortest subpath entirely contained in a single region is composed by at most two edges in the next lower level.

LEMMA 4. *The number of edges in a shortest path expanded to the $(h-i)$-level is $O(2^i)$.*

Proof. If $k > 1$ and defining $c = 2$, any shortest subpath entirely inside a region in a k-level is composed by at most two edges in the $(k-1)$-level. Therefore, the number of edges in a shortest path passing through many regions expanded to the $(k-1)$-level is two times the number of edges in the corresponding subpath in the k-level plus four edges concerning the subpaths originally in the $(k-1)$-level: P^+_{k-1} and P^-_{k-1}. Hence, the number $|P_h|$ of edges in the highest level shortest path is two and the number of edges in a shortest path expanded to the $(h-i)$-level is defined by the recurrence $|P_{h-i}| = 2|P_{h-i+1}| + 4$. This recurrence is evaluated to $|P_{h-i}| = 6(2^i - 1) + 2$. Therefore, the number of edges in a shortest subpath expanded to the $(h-i)$-level is $O(2^i)$. □

For intra-regional queries, we may use any flat strategy. A path view strategy pre-computes all shortest paths in the network for each region. Since $c = 2$, this strategy spends $O(m)$ time to retrieve a shortest path in the lowest level and $O(1)$ time otherwise. The path view strategy requires $O(m^{1.5})$ space in the lowest level and $O(c^k m)$ otherwise.

THEOREM 4. *If the intra-regional query is implemented using a path view approach, the algorithm* `Expand-Path` *spends $O(n)$ time.*

Proof. The traversal of the shortest path tree layers using procedure `Traverse-Backwards` in the lowest level requires $O(m)$ time. Since any subpath is composed by two edges in higher levels, the traversal spends time $O(h) = O(\log r)$ in all higher levels. Therefore, the traversal of the shortest path tree layers takes $O(m + \log r)$ time. Using the path view approach, the procedure `Intra-Regional` spends linear time in the size of the subpath. Hence, the procedure spends $O(m)$ time if $k = 1$, and $O(1)$ time otherwise. Therefore, according to Lemma 4, the subpaths are expanded in time $O(\sum_{k=h}^{2} 2^{h-k} + 2^{h-1}m)$. This expression is equivalent to $O(2^h + 2^{h-1}m) = O(r + rm)$. Therefore, the algorithm `Expand-Path` spends $O(n)$ time. □

The path views 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 5. *If the intra-regional query is implemented using a path view approach, the algorithm* `Expand-Path` *requires $O(n + m^{1.5})$ space.*

Proof. The hierarchical shortest path obtained by the traversal of the shortest path tree layers has $O(m + \log r)$ edges through all levels of the hierarchy. A path view PV_k requires $O(|B_k|^2)$ space in a higher level, since a shortest subpath in a higher level has at most two edges. Hence, a path view requires $O(c^k m)$ space in a higher level while in the lowest level, a path view requires $O(|B_k||V_k|) = O(m^{1.5})$ space. Therefore, the space required to store the path views is $O\left(\sum_{k=1}^{h-1} c^k m + m^{1.5}\right)$. This expression is equivalent to $O(rm + m^{1.5}) = O(n + m^{1.5})$. The expanded path P has $O(2^i)$ edges in the $(h-i)$-level according to Lemma 4. Hence, the space required to store P in the 1-level is $O(2^{h-1}) = O(r)$. In the worst case, each edge of P in the 1-level is expanded into $O(m)$ edges in the lowest level. This way, the path P requires $O(rm) = O(n)$ space. Therefore, the algorithm `Expand-Path` requires $O(n + m^{1.5})$ space in a path view approach. □

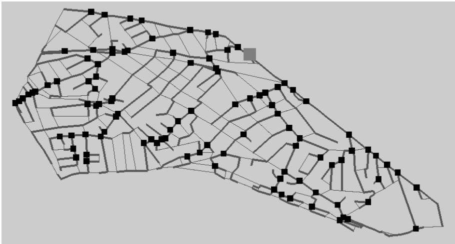
4. THE HYBRID SHORTEST PATH ALGORITHM

Given an edge $e = (u, v)_i$ in the k-level network, an intra-regional query consists in expanding the edge e into a subpath from u to v in the subgraph G^i_{k-1} in 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 between a single-source shortest path algorithm and a lookup in a path view for border vertices.

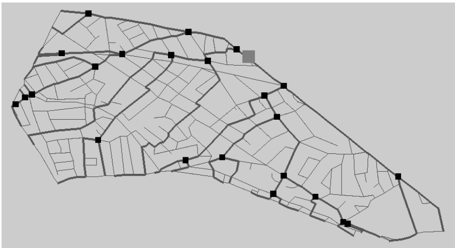
The strategy proposed in this paper for the lowest level uses a hybrid path view for border vertices. The hybrid path view represents each shortest path between border vertices just by a sequence of guide vertices. A *guide vertex* is a predecessor vertex for more than one vertex in the predecessor matrix implementing the path view for border vertices. The path view for border vertices consists of shortest path trees composed only by shortest paths from a border vertex to the other border vertices (see Fig. 6). Each row of a predecessor matrix represents a shortest path tree corresponding to a particular source vertex as the root.

LEMMA 5. *The number of guide vertices in a shortest path tree of a path view for a subgraph in the lowest level is $O(\sqrt{m})$.*

Proof. Concerning the number of guide vertices, the worst case for a shortest path tree is a binary tree. Since a shortest



(a) Destinations are all vertices.



(b) Destinations are only border vertices.

Figure 6: Shortest path trees and guide vertices in a lowest level region.

path tree for a subgraph in the lowest level has $|B_0|$ leaves, there are $|B_0|$ interior nodes that are guide vertices. Therefore, according to Lemma 1, the number of guide vertices in a shortest path tree for a subgraph in the lowest level is $O(\sqrt{m})$. $\square$

The hybrid path view $\overline{PV_0^i}$ for each region i in the lowest level of the hierarchy is retrieved from the corresponding path view PV_0^i. In order to create $\overline{PV_0^i}$, each guide vertex is identified in a traversal of PV_0^i. A hybrid path view is implemented as a predecessor sparse matrix whose columns are only related to guide vertices $\overline{V_0^i}$ and border vertices B_0^i. Further, the predecessor relation in this sparse matrix is only expressed in terms of guide vertices.

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 lowest 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 $V_0^i - PT$. 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.

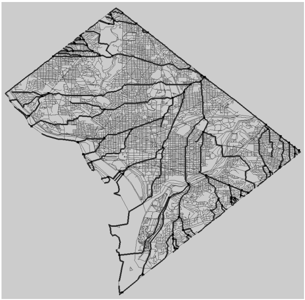
Algorithm `Hybrid-Shortest-Path`(u, v, i)

1. $\overline{P_0^i}(u, v) \leftarrow$ `Path-Lookup`$\left(\overline{PV_0^i}, u, v\right)$
2. $PT \leftarrow Q \leftarrow \emptyset$
3. $Q \leftarrow Q' \leftarrow Q \cup (u, 0, 0)$
4. **for** $v' \in V_0^i - u$ **do**
   (a) $Q \leftarrow Q \cup (v', \infty, \infty)$
5. **for** each $(s, d) \in \overline{P_0^i}(u, v)$ **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** $(u', u'')_j \in E_0^i$
         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** $(d, u'')_j \in E_0^i$
      i. $f(u'') \leftarrow |(d, u'')_j|$
      ii. $e'' \leftarrow (d, u'')_j$
   (g) $Q' \leftarrow \emptyset$
6. $P \leftarrow$`TravBack`(PT, v)

First, the algorithm finds the partial shortest path $\overline{P_0^i}(u, v)$ using the procedure `Path-Lookup` that traverses the hybrid path view $\overline{PV_0^i}$ (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). Then, the algorithm finds a shortest subpath in G_0^i 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 following 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 `TravBack` (step 6).

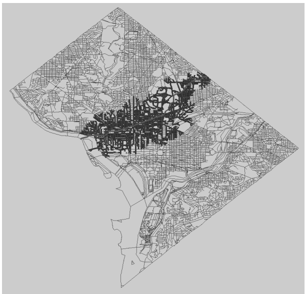
In the worst case, the hybrid shortest path algorithm has the same time requirement as Dijkstra's single-source shortest path algorithm. However, the number of vertices visited by the hybrid shortest path algorithm is much smaller than all vertices when vertices are uniformly distributed in the region corresponding to the subgraph and the guide vertices are uniformly distributed along a shortest path (see Fig. 7).

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 requirement for the algorithm is the same as for a single-source shortest path algorithm.



(a) Shortest path tree for border vertices.

(b) Dijkstra's algorithm.



(c) Hybrid algorithm.

Figure 7: Vertices visited by the shortest path algorithms.

THEOREM 6. *The hybrid shortest path algorithm requires $O(m)$ space.*

Proof. The hybrid path view $\overline{PV_0^i}$ requires $O(|B_0|\overline{V_0})$. Hence, according to Lemmas 1 and 5,$\overline{PV_0^i}$ requires $O(m)$ space. The partial path $\overline{P_0^i}$ requires $O(|\overline{V_0}|+|B_0|) = O(\sqrt{m})$ space. The space required by the set PT, priority queue Q and set Q' is $O(|V_0|)$. Hence, PT, Q and Q' require $O(m)$ space. The set of edges E_0 requires $O(m)$ space. Therefore, the hybrid shortest path algorithm requires $O(m)$ space. $\square$

An uniform distribution of vertices in the lowest level is acomplished when the number of vertices m in a bounded region tends to infinity and the closest pair of points has a distance greater than some small constant. This way, the shortest path tends to a straight line and the number of vertices in a shortest path is $O(\sqrt{m})$.

THEOREM 7. *Assume an uniform distribution of vertices in 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 $O(\sqrt{m})$.*

Proof. Assuming the guide vertices are uniformly distributed along the shortest path, the number q of vertices between two consecutive guide vertices in a shortest path is $O\left(\frac{\sqrt{m}}{|\overline{V_0}|}\right)$. The number of vertices visited in a single search for a shortest subpath between two consecutive guide vertices is $O(q^2)$. This way, the total number of vertices visited is $O\left(|\overline{V_0}|q^2\right)$. This expression is equivalent to $O\left(\frac{m}{|\overline{V_0}|}\right) = O\left(\frac{m}{\sqrt{m}}\right)$. Therefore, the number of vertices visited by the hybrid shortest path algorithm is $O(\sqrt{m})$. $\square$

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

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

Proof. The path lookup in the hybrid path view spends time $O(\sqrt{m})$. The priority queue initialization requires $O(m)$ time. According to Lemma 7, the number of vertices visited by the algorithm is $O(\sqrt{m})$. This way, the algorithm spends $O(\sqrt{m}\log m)$ time in `Extract-Min` operations and cost updates. The algorithm spends $O(\sqrt{m})$ time in resets and initializing Q concerning source vertices. Therefore, the algorithm spends $O(\sqrt{m}\log m + m)$ time. Since $\log m = O(\sqrt{m})$, the hybrid shortest path algorithm spends $O(m)$ time in the lowest level. $\square$

The best time and space requirements for intra-regional queries in the lowest level is accomplished by the hybrid shortest path algorithm assuming an uniform distribution of vertices. For higher levels, the best strategy concerning time and space resources is the path view approach defining $c = 2$.

5. CONCLUSION AND FUTURE WORK

We present a hierarchical approach that subdivides the network into regions with the same number of vertices and creates higher levels merging a constant number of adjacent regions. The hierarchy is completely defined by the number of regions r in the lowest level, the number m of vertices of a region in the lowest level, and the merging degree c. We propose algorithms to find a shortest path and to expand this path.

The complexity analysis for these algorithms show that our hierarchical approach requires $O(n)$ query time and space when $r = m = \sqrt{n}$ and $c = 2$. Since our approach is nonrecursive, the complexity constants are small. Hence, it is more efficient in practice when compared to other recursive optimal approaches.

A hybrid shortest path algorithm to perform intra-regional queries is introduced. This strategy uses a subsequence of vertices that belong to the shortest path while actually computing the whole shortest path. In the lowest level, the hybrid algorithm requires $O(m)$ time and space assuming an uniform distribution of vertices. This is better than a path view approach that requires $O(m^{1.5})$ space. For higher levels, the path view approach spends $O(1)$ time and requires $O(c^k m)$ space.

The algorithms in the hierarchical approach also present a good potential for parallelization. The path expansion may expand each subpath for a specific level in parallel.

In order to support the theoretical results we achieved, computational experiments using real networks to evaluate the time and space efficiency for our hierarchical approach should be implemented. Three different strategies to perform intra-regional queries in the lowest level would be compared: Dijkstra's algorithm, path view lookup and hybrid shortest path algorithm.

6. REFERENCES

[1] D. Chen and J. Xu. Shortest path queries in planar graphs. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 469–478. ACM, May 2000.

[2] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3):505–536, July 1985.

[3] E. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[4] H. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *WG: Graph-Theoretic Concepts in Computer Science, International Workshop WG*, 1996.

[5] G. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.*, 16(6):1004–1022, December 1987.

[6] G. Frederickson. Planar graph decomposition and all pairs shortest paths. *Journal of the ACM*, 38(1):162–204, January 1991.

[7] M. Fredman and R. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, July 1987.

[8] M. Goodrich. Planar separators and parallel polygon triangulation. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 507–516. ACM, May 1992.

[9] N. Jing, Y.-W. Huang, and E. Rundensteiner. 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] D. Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM*, 24(1):1–13, January 1977.

[11] P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 27–37. ACM, May 1994.

[12] R. Lipton and R. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math*, 36:177–189, 1979.

[13] S. Shekhar, A. Fetterer, and B. Goyal. Materialization trade-offs in hierarchical shortest path algorithms. In M. Scholl and A. Voisard, editors, *SSD'97, Proceedings of the 5th International Symposium on Advances in Spatial Databases*, volume 1262 of *Lecture Notes in Computer Science*, pages 94–111. Springer-Verlag, 1997.