IMPROVING PATH QUERY PERFORMANCE IN PGROUTING USING A MAP GENERALIZATION APPROACH

Rohith Reddy Sankepally, K.S Rajan

Lab for Spatial Informatics, International Institute of Information Technology, Gachibowli, Hyderabad, India - (rohith.reddy, rajan)@iiit.ac.in

Commission V, WG V/8

KEY WORDS: Pgrouting, Road Networks, Skeletal Model, Skeleton, Shortest Path, Map Generalization, Zones, Path Computation

ABSTRACT:

pgRouting library provides functions to compute shortest path between any two points of a road network which is of great demand and also a topic of interest in the field of GIS, graph theory and transportation. To compute path in a road network, pgRouting functions process the entire road network which is a major bottleneck when it comes to routing in large road networks leading to the requirement of large server resources. A reduction/compression in the input network that is to be processed for path computation would improve the performance of pgRouting. In this study a map generalization based network model is proposed which extracts a significantly smaller subset of the road network aka *skeleton* which further used to divide the network into *zones*, that shall be selectively used in path computation. This results in processing a much smaller part of the network to compute path between any two points leading to an overall improvement in query performance of pgRouting when computing path, especially on large road networks. As part of assessment of this approach and its applicability to large road networks, the paper presents an in-depth analysis of the trade-offs between deviation in computed path and the performance gain in terms of space and time on road networks of varying sizes and topology to get a better understanding for both providing a sound proof of the utility of the proposed method and also to show its implementability within the current model of pgRouting or any other routing platforms.

1. INTRODUCTION

pgRouting is an open source geospatial routing library extending PostGIS enabled PostgreSQL database. pgRouting library provides a variety of routing algorithms like All Pairs Shortest Path(APSP), Shortest Path, Driving Distance, Traveling Sales Person and Turn Restricted Shortest Path(TRSP). These routing algorithms are of great demand and a topic of interest in the field of GIS, graph theory and transportation. pgRouting path algorithms process the full network to compute path between any two points. This leads to slow path computation especially in case of large road networks. A reduction/compression in network data processed for path computation should enhance the performance of pgRouting path algorithms. A number of approaches have been tried out in this context of network reduction, like network compression, graph contraction, graph partitioning and map generalization which are discussed below.

(Akimov et al., 2004, Khoshgozaran et al., 2008, Shekhar et al., 2002, Suh et al., 2007, Zhang, 2006) try to compress the vector data or road networks. (Akimov et al., 2004) deals with compression of vector data by removing redundancy in the data. (Khoshgozaran et al., 2008) implements a compression technique that improves performance of vector data queries. (Suh et al., 2007, Shekhar et al., 2002) propose techniques of vector data compression which can be used to reduce storage and improve data transportation in limited bandwidth. Most of these works talk about compressing vector/road network data but do not deal with path computation on the compressed network which may lead to improved performance in path computation.

(Geisberger et al., 2008) try to contract the graph by addition of shortcuts and store precomputed paths to achieve speedups in path computation. (Möhring et al., 2005, Jung and Pramanik, 1996, Chondrogiannis and Gamper, 2016) try to partition the graph into clusters and store precomputed paths to reduce the search space and improve path computation. In each of the above mentioned works, the graph is modified due to the addition of shortcuts. The precomputed paths need to be computed and stored which leads to additional storage requirements. Moreover the path extracted using the modified graph is not complete and needs expansion due to presence of shortcuts in the path. This maybe an overhead when computing longer paths in large road networks. Some of these works also require the design of a special algorithm to compute path based on the modified graph structure.

In literature, several generalization approaches for road networks have been proposed. (Thomson and Richardson, 1995, Mackaness and Beard, 1993, Jiang and Claramunt, 2004, Jiang and Harrie, 2004) propose *graph theory* based generalization methods. (Bjørke and Isaksen, 2005) deals with the applicability of *information theory* to generalization. (Thomson and Richardson, 1999) proposes a *topography* based generalization approach. In all the above studies focus has been mainly on generating a generalized network with reduced size while preserving its overall topography. Not much work has been carried out on exploiting this generalized structure of the network which may achieve gains in path computation.

The goal of this study is to improve the overall query performance of pgRouting by proposing a map generalization based network model that leads to processing a significantly small subset of the road network selectively, to compute path between any two points without the use of precomputed paths. The proposed approach is evaluated by carrying out an in-depth analysis of the trade-offs between deviation in computed path and the performance gain in terms of space and time on road networks of varying sizes and topology. It should be noted that the none of the above mentioned approaches in the literature are implemented in pgRouting

for path computation. Therefore the proposed approach is compared to the pgRouting Dijkstra algorithm to get a better understanding of the utility of the proposed method and also to show its implementability within the current architecture of pgRouting or any other routing platforms.

2. NOTATIONS

2.1 Graphs and Paths

A road network can be represented as a directed graph $G = (V, E)$, where V denotes a set of nodes that represent road intersections and $E \subseteq V \times V$ is the set of edges. Each edge $e = (v_a, v_b)$, represents a road segment that connects nodes v_a and v_b. A weight function w: $E \rightarrow R$ assigns to each edge $e = (v_a, v_b)$ a weight $w(e)$, which captures the cost of moving from v_a to v_b, in terms of travel time or distance. A *path* p is an ordered set of edges $e_1, e_2, ..., e_N$ where $e_i \in E$ is an edge $\forall i$ where $i = 1, 2, ..., N$. A *path* between nodes v_x and v_y is denoted by $p(x \rightarrow y)$. The containment of an edge e in a path p defined by a function σ as follows

$$\sigma(p(x \rightarrow y), e) = \begin{cases} 1, & \text{if } e \in p(x \rightarrow y). \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

The length $l(p)$ of a path p equals the sum of the weights for all contained edges, i.e.,

$$l(p) = \sum_{i=1}^{N} w(e_i). \quad (2)$$

$p^*(x \rightarrow y)$ is a shortest path if there is no path $p(x \rightarrow y)$ such that $l(p) < l(p^*)$.

2.2 Subgraphs and Connectivity

A directed graph is said to be *connected* if every node is reachable from every other node i.e a path p exists between each and every pair of the nodes. A *connected component* is a subgraph in which any two nodes are connected to each other by paths, and is connected to no additional node in its corresponding supergraph. Suppose a graph $G = (V, E)$ is divided into a set of subgraphs $\{SG_1(V_1, E_1), SG_2(V_2, E_2), ..., SG_n(V_n, E_n)\}$ then,

$$V_1 \cup V_2 \cup .... \cup V_n = V,\ E_1 \cup E_2 \cup .... \cup E_n \subset E$$
$$V_i \cap V_j = \varphi,\ E_i \cap E_j = \varphi$$
$$where\ 1 \leq i, j \leq n and\ i \neq j$$

A set of *connecting edges* E_{conn} for a graph $G(V, E)$ is the set of all edges (v_a, v_b) such that v_a and v_b belong to two subgraphs SG_a and SG_b respectively where $a \neq b$.

$$E_1 \cup E_2 \cup .... \cup E_n \cup E_{conn} = E \quad (3)$$

For each subgraph $SG_j(V_j, E_j)$, a set of connecting edges $E_{conn}(SG_j)$ is defined as

$$E_{conn}(SG_j) = \{(v_a, v_b) | v_a \in V_j \wedge v_b \notin V_j)\}$$
$$E_{conn}(SG_j) \subset E_{conn} \quad \forall j, 1 \leq j \leq n$$

The above definitions can be generalized into multiple levels. We use the notation $SG_j^i(V_j^i, E_j^i)$ to denote the *jth* subgraph in the *ith* level, and $E_{conn}(SG_j^i)$ denotes the set of connecting edges of $SG_j^i(V_j^i, E_j^i)$. E_{conn}^i as the set of all *connecting edges* in the *ith* level.

2.3 Skeleton Network

A *Skeleton Network* $G_s = (V_s, E_s)$ is a *connected component* of $G = (V, E)$. The *Skeleton Network* is a representative network of the original network whose size is very less compared to the original network. The above definition of G_s can be generalized into multiple levels. We use the notation G_s^i to denote the *Skeleton Network* in the *ith* level.

2.4 Residual Network

A *Residual Network* $G_{res} = (V_{res}, E_{res})$ of a *Skeleton Network* G_s in $G = (V, E)$ such that

$$V_{res} = V - V_s$$
$$E_{res} = E - E_s - E_{conn}(G_s)$$

The above definition can be generalized into multiple levels. We use the notation $G_{res}^i = (V_{res}^i, E_{res}^i)$ to denote the *Residual Network* of $G_s^i = (V_s^i, E_s^i)$ in the *ith* level.

3. ARCHITECTURE OF PGROUTING

pgRouting follows an SQL based architecture in which the graph/network data is stored in the form of SQL tables in a PostgreSQL database. The graph data for path computation is extracted quickly and efficiently using SQL queries. Figure 1 shows the architecture of pgRouting comprising of two major components namely *PostgreSQL Database* and the *pgRouting extension* which are explained below.

3.1 PostgreSQL Database

The postgreSQL database contains the information of edges and vertices of the graph/network $G(V, E)$ in the form of SQL tables. The schema for edge table is explained in Table 1.

Table 1. Table Schema of Edges

Column	Data Type	Description
id	long int	A unique identifier assigned to every edge.
source	long int	Identifier for the source node of the edge.
target	long int	Identifier for the target node of the edge.
cost	real	The length of the edge
the_geom	geometry	A postGIS attribute which represents the geometry of the edge.

3.2 pgRouting Extension

pgRouting is an extension to the PostgreSQL database which contains all the path computation algorithms. Let us say the client wants to find a path between source node v_x and target node v_y. The client sends a request in the form of an SQL query to the PostgreSQL server to compute a path between v_x and v_y by specifying a path algorithm, let us say P_A. The pgRouting extension extracts the appropriate graph data from the edge table in the PostgreSQL database using SQL query. The extracted graph data is now used to find the path between v_x and v_y by using P_A. The computed path is then returned back to the client.

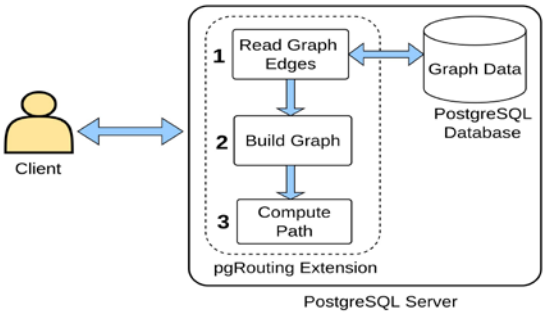


Figure 1. pgRouting Architecture.

3.2.1 Query Structure Given below is the signature for pgRouting Dijkstra Algorithm.

```
pgr_dijkstra(TEXT edges_sql,
BIGINT start_vid, BIGINT end_vid)
RETURNS SET OF (seq, path_seq,
node, edge, cost, agg_cost)
or EMPTY SET
```

Let us try to understand the signature with a sample query given below

```
SELECT * FROM pgr_dijkstra(
SELECT id,source,target,cost
FROM edge_table,
4, 2);
```

The mapping between the algorithm signature and sample query is shown in Table 2

Table 2. Mapping between Signature and Query

Signature	Query
edges_sql	SELECT id,source,target,cost FROM edge_table
start_vid	*4*
end_vid	*2*

The *edges_sql* represents the edges of the graph G on which the path computation is performed. *start_vid* represents the identifier of the source v_x. *end_vid* represents the identifier of the target v_y.

3.2.2 Path Computation The path computation procedure can be divided into 3 simple steps as illustrated in Figure 1.

1. **Read Graph Edges** In this step the function extracts the graph specified by the client in the query. The graph corresponding to *edges_sql* query is obtained from the edge table present in the PostGreSQL database.
2. **Build Graph** This step uses the graph extracted from Step 1 to build a Boost C++ graph structure internally.
3. **Compute Path** In this step path between source and target is computed by using the specified path algorithm on the internal C++ graph structure obtained from Step 2.

The pgRouting function in the server takes v_x and v_y as input and executes steps 1, 2 and 3 to compute path between them and returns the path to the client. The output format of the pgRouting path algorithm is given in Table 3. The output of a dijkstra query to compute path from source node 4 to target node 2 in the graph(shown in Figure 2(a) is given below.

Table 3. Output Format of pgRouting Dijkstra Query

Column	Data Type	Description
seq	INT	Sequential value starting from 1.
path_seq	INT	Relative position in the path. Has value 1 for the beginning of a path.
start_vid	BIGINT	Identifier of the starting vertex.
end_vid	BIGINT	Identifier of the ending vertex.
node	BIGINT	Identifier of the node in the path from start_vid to end_vid.
edge	BIGINT	Identifier of the edge used to go from node to the next node in path sequence. -1 indicates the last node of the path.
cost	FLOAT	Cost to traverse from node using edge to the next node in the path sequence.
agg_cost	FLOAT	Aggregate cost from start_vid to node.

```
SELECT * FROM pgr_dijkstra(
    SELECT id, source, target,
    cost FROM edge_table,
    4, 2
);
 seq|path_seq|node|edge|cost|agg_cost
-- --+--------+----+----+----+--------
  1 |      1 |  4 |  3 |  1 |      0
  2 |      2 |  3 |  2 |  1 |      1
  3 |      3 |  2 | -1 |  0 |      2
(3 rows)
```

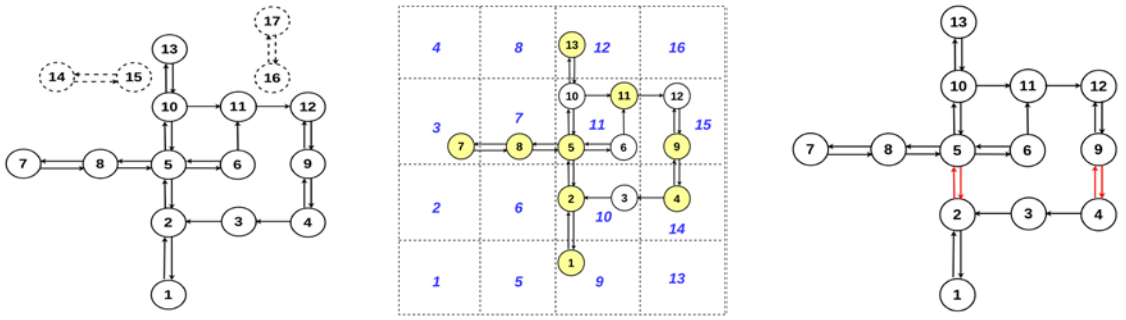
To compute path between any two points in a network, pgRouting path algorithms use the full network which leads to performance issues when it comes to routing in large road networks. To improve path computation either the individual steps explained in 3.2.2 or their combination needs to be enhanced. In this work we focus on improving the efficiency of step (1) by a reduction or compression in the extracted network data for path computation, which naturally leads to an improvement in steps (2) & (3).

4. SKELETAL MODEL : A NETWORK GENERALIZATION APPROACH

4.1 Preprocessing

Given a graph $G = (V, E)$, all the disconnectivities and dangles are removed and it is ensured that the graph is well *connected*. Figure 2 (a) illustrates a graph $G = (V, E)$. The disconnectivities and dangles are represented by dotted lines. After preprocessing, the dangles are removed and the resulting graph is shown in Figure 2 (b). All the edges of $G = (V, E)$ shown in Figure 2(b) have unit weight.

4.2 Edge Priority



(a) Graph with disconnectivities. (b) Graph after preprocessing. (c) Edges with $c(e) \geq 28$

Figure 2. Computation of Edge Priorities

4.2.1 Grid based Division The graph $G(V, E)$ is divided into $g \times g$ grids as shown in Fig 2 (b). Every grid is assigned an identifier i.e *grid_id*. Each vertex is populated with a *grid_id* indicating the grid to which it belongs. Let V^j be the set of nodes that belong to the *jth* grid such that

$$V = \bigcup_{j=1}^{g^2} V^j$$

4.2.2 Choice of Nodes After the network G was divided into grids, nodes that belong to each grid are known. From each grid j a set of random nodes V^j_{noi} were chosen. The number of nodes chosen from each grid is proportional to the number of nodes that belong to the grid. The collection of chosen nodes from all the grids constitute the *special nodes of interest*.

$$V_{noi} = \bigcup_{j=1}^{g^2} V^j_{noi}$$
$$where\ V^j_{noi} \subset V^j \quad \forall j, 1 \leq j \leq g^2$$

Figure 2(b) shows that the graph $G(V, E)$ is divided into 4×4 grids. The yellow colored nodes constitute the *special nodes of interest*. The identifier of each grid is represented in blue color. The number of nodes chosen from each grid $|V^j_{noi}| = |V^j|^p$ where p = 0.5. The grids $1, 2, 4, 5, 6, 8, 13, 16$ are empty and thus do not contribute towards the *special nodes of interest*.

4.2.3 Priority Definition Dijkstra's shortest path algorithm was used to compute the path between every special node of interest to every other special node of interest. Each edge e is assigned a value $c(e)$ which indicates the number of shortest paths that contain e. This value $c(e)$ determines the importance/priority of an edge e.

$$c(e) = \bigcup_{v_x \in V_{noi}} \bigcup_{v_y \in V_{noi}} \sigma(p^*(x \rightarrow y), e).$$

Using the above definition, the priorities of all the edges are computed. The red colored edges in Figure 2(c) shows the edges with $c(e) \geq 28$.

4.3 Skeleton Construction

4.3.1 Edge Selection The *Skeleton Network* is generated based on the edge priorities $c(e)$ computed in the previous step. In order to have control on the size of the *Skeleton Network* that is supposed to be generated, a threshold value $c_{threshold}$ is chosen. The $c_{threshold}$ value eases the selection of edges based on their priorities. Let us suppose we want to select the top 10 edges of a network based on priorities. In order to do so we sort the edges according to the priorities and choose the top 10 among them. Let us say edge e^t is the 10th edge which is of least priority with priority value $c(e^t)$. In such a case, the top 10 edges can be represented by defining a threshold value $c_{threshold} = c(e^t)$. Now in order to easily extract the top 10 edges from the network, we choose the edges with priority value greater than $c_{threshold}$. G_s is initialized with an empty set and the edges $e \in E$ with $c(e) \geq c_{threshold}$ are added to G_s.

4.3.2 Connectivity The resultant G_s formed in the previous step, may not be *connected*. According to the definition of *Skeleton Network* as given in Section 2.3 it is a well *connected* subgraph. To satisfy the connectivity constraint the graph G_s formed in the previous step should be made well connected to be qualified as the *Skeleton Network*. In order to make G_s connected the following steps are performed in order

1. Find the well connected components of G_s
2. Add paths between the connected components until G_s becomes well connected.

The above mentioned steps are performed and the resultant G_s is as shown in Figure 3 (a). Algorithm 1 is used to generate a Skeleton Network given a threshold value $c_{threshold}$. Algorithm 2 explains the procedure to make G_s well connected.

Algorithm 1 Value Based Skeleton Network Construction Algorithm

```
1: procedure ConstructSkeleton(G, c_threshold)
2:     G_s ← φ
3:     for e ∈ E do
4:         if c(e) ≥ c_threshold then G_s.add_edge(e)
5:     MakeConnected(G_s)
6:     return G_s
```

4.4 Zone Generation

Definition 1. The residual network G_{res}(defined in Section 2.4) is partitioned into a set of mutually disconnected subgraphs $\{Z_1(V_{z1}, E_{z1}), Z_2(V_{z2}, E_{z2}), \ldots, Z_n(V_{zn}, E_{zn})\}$ called *zones* such that

$$G_{res} = \bigcup_{j=1}^{n} Z_j \quad (4)$$

Algorithm 2 Connected Skeleton Algorithm

```
1: procedure MAKECONNECTED(G_s)
2:     {SG_1, ..., SG_n} ← getConnectedComponents(G_s)
3:     if n /= 1 then
4:         Let v_a ∈ SG_a, v_b ∈ SG_b where a /= b
5:         for e ∈ p(v_a → v_b) do
6:             if e /∈ E_s then
7:                 G_s.add_edge(e)
8:         MakeConnected(G_s)
9:     else
10:        return G_s
```

Definition 2. For each zone Z_j, a set of *skeletal nodes* $S(Z_j)$ is defined as

$$S(Z_j) = \{v | (u,v) \in E_{conn} \wedge u \in V_{zj} \wedge v \in V_s\} \cup \{u | (u,v) \in E_{conn} \wedge v \in V_{zj} \wedge u \in V_s\}$$

Definition 3. Given a zone $Z_j(V_{zj}, E_{zj})$, its *extended zone* is defined as a subgraph $Z_j^*(V_{zj}^*, E_{zj}^*)$ such that

$$V_{zj}^* = V_{zj} \cup S(Z_j)$$
$$E_{zj}^* = E_{zj} \cup E_{conn}(Z_j)$$

Definition 4. Two extended zones $Z_a^*(V_{za}^*, E_{za}^*)$ and $Z_b^*(V_{zb}, E_{zb})$ are connected only when

$$S(Z_a) \cap S(Z_b) /= \varphi, a /= b$$

Given the skeleton G_s, the corresponding residual network G_{res} obtained and all the *zones* of G_{res} are computed. It should be noted that each of the zones Z_j are well connected to the skeleton G_s through a set of connecting edges $E_{conn}(Z_j)$. The above definitions can be generalized into multiple levels. Z_j^i denotes ith *zone* in the ith level. $S(Z_j^i)$ denotes the set of skeletal nodes of Z_j^i. $E_{conn}(Z_j^i)$ denotes the set of connecting edges of Z_j^i. $Z_j^*(i)$ denotes the *extended zone* of Z_j^i.

Figure 3(b) shows the zones Z_1, Z_2 and Z_3 formed as a result of partitioning G_{res}. The dashed edges connecting the *skeleton* and the *zones* represent the connecting edges E_{conn} defined in Section 2.2.

4.5 Edge Levels

In order to generate skeletons of different sizes, a quantile classification with k intervals, is applied over $c(e)$ values sorted in decreasing order to get a qualitative classification of edges based on their priority in the overall network. Each interval i is associated with $c_{threshold}^i$ which denote the minimum priority value of interval i where $1 \leq i \leq k$. Every interval is termed as a *level* such that the ith interval denotes the ith level. Every edge e is assigned a value $level(e)$ which denotes its priority level. Algorithm 3 is used to generate skeleton at a given level i where $1 \leq i \leq k$.

$$level(e) = \{ min(i) \mid c(e) \geq c_{threshold}^i \}$$
$$1 \leq level(e) \leq k$$

Algorithm 3 Level Based Skeleton Construction Algorithm

```
1: procedure CONSTRUCTSKELETON(G, i)
2:     G_s^i ← φ
3:     for e ∈ E do
4:         if level(e) ≤ i then G_s^i.add_edge(e)
5:     MakeConnected(G_s^i)
6:     return G_s^i
```

5. PGROUTING PATH ALGORITHM BASED ON SKELETAL MODEL

The skeleton of a road network can be used to optimize path computation by limiting the amount of network that is used for path computation which could solve the bottleneck problem of pgRouting and improve the performance of path computation. This section talks about the applicability of the Skeletal Model to the existing pgRouting Architecture to improve path computation.

After the formation of zones as discussed in Section 4.4, zone identifier is assigned to all edges and nodes of G_{res} indicating the *zone* to which they belong. $z(e)$ and $z(v)$ indicate the zone to which edge e and vertex v belong to respectively. We use the notation $z^i(e)$ and $z^i(v)$ to denote the *zone identifier* of edge e and node v respectively in the ith level.

Definition 5. Let v_x be the source node and v_y be the target node between which the path is to be computed. Let $Z_{x'}$ and $Z_{y'}$ be the *zones* to which v_x and v_y belong respectively where $x' = z(v_x)$, $y' = z(v_y)$ and $x /= y$. Let $G_s = (V_s, E_s)$ be the *Skeleton Network* of $G = (V, E)$. The path between v_x and v_y is computed on the *reduced* graph $G_{x,y}$ where

$$G_{x,y} = Z_{x'}^* \cup G_s \cup Z_{y'}^* \quad (5)$$

The above definition of $G_{x,y}$ can be generalized to multiple levels where $G_{x,y}^i$ denotes the *reduced* graph in the ith level given source node v_x and target node v_y.

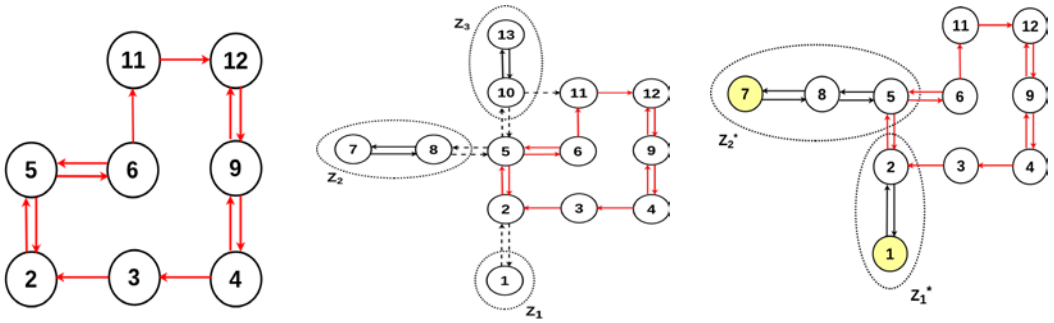
5.1 Path Computation Algorithm

The reduced graph concept explained in the previous section is used for computing path between any two nodes in the network. From Section 4.4 it is ensured that the every extended zone is well connected to the skeleton G_s. From Section 2.3, skeleton G_s is also well connected. Therefore it can be deduced that the reduced graph $G_{x,y}$ is also well connected and thus contains the path between any two nodes v_x and v_y. Algorithm 4 illustrates the algorithm that computes path between a source node v_x and target node v_y using the reduced graph.

Algorithm 4 Path Computation Algorithm

```
1: procedure SKELETALPATHALGORITHM(v_x, v_y)
2:     x' = z(v_x)
3:     y' = z(v_y)
4:     G_{x,y} = Z*_{x'} ∪ G_s ∪ Z*_{y'}
5:     p = DijsktraAlgorithm(G_{x,y}, v_x, v_y)
6:     return p
```

Figure 3(c) illustrates the reduced graph $G_{x,y}$ i.e $G_{1,7}$ used for path computation algorithm to compute path from source node 1 to target node 7 with $x' = z(1) = 1$ and $y' = z(7) = 2$. The red colored edges represent the skeletal edges E_s and the graph enclosed within the dotted lines represent the extended zones Z_1^*



(a) Skeleton Network. (b) Zones (c) Reduced Graph

Figure 3. Path Computation Model

and Z_2^* to which node 1 node 7 belong to respectively. From Figure 3(c) it can be noticed that the reduced graph $G_{1,7}$ does not contain the extended zone Z_3^* and therefore a reduction in the graph size can be observed which cuts down the overall search space for path algorithms.

5.2 pgRouting Skeleton Path Query

The proposed path algorithm reveals that only a subset of the graph is sufficient to compute path between any two points. This algorithm when implemented in pgRouting reduces the overhead of extracting and processing the total road network for path computation thus solving the bottleneck problem of pgRouting. Moreover it can be implemented in pgRouting easily by reusing the existing pgRouting path algorithm without any changes to the existing architecture. k new columns are added to the edge table and vertex table in the PostgreSQL database to store the zone identifier of each edge e and vertex v respectively in the ith level. The columns are named as $zone\_i$ which represents the value $z^i(e)$ for each edge and $z^i(v)$ for each vertex where $1 \leq i \leq k$. The equations below give a clear understanding of the assignment of zone identifier values to each edge e and node v in the ith level.

$$z^i(v_a, v_b) = \begin{cases} 0, & v_a \in V_s^i \,\&\, v_b \in V_s^i. \\ j, & v_a \in V_j^i \,\&\, v_b \in V_j^i. \\ -j, & v_a \in V_j^i \,\&\, v_b \in V_s^i. \\ -j, & v_a \in V_s^i \,\&\, v_b \in V_j^i. \end{cases} \quad (6)$$

$$z^i(v_a) = \begin{cases} 0, & v_a \in V_s^i. \\ j, & v_a \in V_j^i. \end{cases} \quad (7)$$

In order to easily extract the residual graph $G_{x,y}$ in the SQL based architecture the connecting edges $e \in E_{conn}$ connecting G_s and Z_j are given a value $-j$.

Given this configuration of column names and zone identifier assignment we try to understand how different components of the reduced graph $G_{x,y}^i$ namely $G_s^i, Z_{x^l}^*(i) \,\&\, Z_{y^l}^*(i)$ at the ith level where $x^t = z^i(v_x)$ and $y^t = z^i(v_y)$, are extracted easily using SQL queries. Figure 3(c) shows the reduced graph used for understanding such queries at level $i = 1$ where $x = 1$ and $y = 7$ and their corresponding zone identifiers $x^t = z^i(1) = 1$ and $y^t = z^i(7) = 2$.

The skeleton G_s^1 is extracted using the query below

```
SELECT id,source,target,cost
FROM edge_table WHERE zone_1 = 0;
```

The extended component $Z_1^*(1)$ is extracted using the query below. The ABS function in SQL is mathematical function that returns the absolute (positive) value of the specified numeric expression.

```
SELECT id,source,target,cost
FROM edge_table WHERE ABS(zone_1) = 1;
```

Similarly, the extended component $Z_2^*(1)$ is extracted using the query below

```
SELECT id,source,target,cost
FROM edge_table WHERE ABS(zone_1) = 2;
```

Combining the above three individual queries the query to extract the reduced graph $G_{1,7}$ is given by the following query

```
SELECT id,source,target,cost
FROM edge_table WHERE zone_1 = 0
OR ABS(zone_1) = 1 OR ABS(zone_1) = 2;
```

Therefore the path query can be written as

```
SELECT * FROM pgr_dijkstra(
    SELECT id,source,target,cost
    FROM edge_table WHERE zone_1 = 0
    OR ABS(zone_1) = 1 OR ABS(zone_1) = 2,
    1, 7
);
```

seq	path_seq	node	edge	cost	agg_cost
1	1	1	1	1	0
2	2	2	4	1	1
3	3	5	7	1	2
4	4	8	6	1	3
5	5	7	-1	0	4

(5 rows)

Algorithm 5 pgRouting Skeletal Path Query

```
1: procedure PGR_DIJKSTRA(G, v_x, v_y, i)
2:     x^t = z^i(v_x)
3:     y^t = z^i(v_y)
4:     G^i_{x,y} = Z*_{x^l}(i) ∪ G^i_s ∪ Z*_{y^l}(i)
5:     p = DijsktraAlgorithm(G^i_{x,y}, x, y)
6:     return p
```

Algorithm 5 illustrates the level based path computation algorithm implemented in pgRouting. The performance of network extraction from the PostgreSQL can further be improved by creating indices on the added zone columns $zone\_i$ where $1 \leq i \leq k$.

Table 4. Dataset Details

Dataset	V	E	d_{max}	*Preprocessing(min)*
Chandigarh	14675	38066	24 kms	2.3
Hyderabad	206174	527559	90 kms	54.4
NYC	398440	991186	165kms	178.8
Belgium	878720	1914674	483kms	475.8

The proposed model also provides a structured way of network storage in the database leading to efficient retrieval of a subset of network data for path computation.

6. EXPERIMENTS

The road network data of Chandigarh, Hyderabad, NYC and Belgium (see Table 4) made available by Open Street Maps is used for experiments. The road networks required for the analysis were extracted by using *osm2pgrouting* tool. The experiments were carried out on a 64-bit linux machine with an Intel Xeon Z400 equipped with 16 GB main memory and 8 MB L3 cache.

The experimental evaluation is divided into two sections. In the first section we evaluate the preprocessing time, skeleton sizes and extended zone sizes for $k = 10$. In the second section we evaluate the proposed pgRouting path computation algorithm that uses the *Skeletal Model*. For path evaluation we average the path computation time and path error over a set of 1000 randomly generated queries with varying path lengths. The path error we refer to is the difference in the length of the path computed using Algorithm 5 and the length of the same path computed using pgRouting Dijkstra Algorithm that uses the entire network. We divide the set of queries into 5 sets of equal size. The node pairs are generated in such a way that the distance between a node pair in the qth set lies between $\frac{(q-1) \times d_{max}}{5}$ and $\frac{q \times d_{max}}{5}$ where d_{max} is the distance between the farthest node pair in the network and $1 \leq q \leq 5$. Therefore distance between any pair of nodes in $(q+1)th$ set is less than distance between any pair of nodes in qth set. Table 4 contains the value of d_{max} for each of the road networks. Table 4 also shows the combined processing times for calculating edge levels, skeleton generation and zone generation for all the 10 levels.

Chandigarh is a uniform gridded network whereas *Hyderabad* is a non uniform dense network. In order to highlight the applicability to different types of networks, the proposed method is applied to *Chandigarh* and *Hyderabad* road networks and the observations are explained. In order to highlight the applicability to large networks, the proposed method is applied to *NYC* and *Belgium* road networks and the observations are tabulated in Table 5.

Figure 4(a) and 5(a), shows the size of skeleton network at each level i generated as a percentage of the original network $G(V, E)$. Here by size we refer to the number of edges in the graph. We can observe that the size of the skeleton increases as level i increases. This is because as the level increases more and more edges are added to the skeleton as discussed in Section 4.5.

Figure 4(b) and 5(b), shows the average and maximum size of the *extended zones* generated at each level i as a percentage of the original graph $G(V, E)$. Here by size we refer to the number of edges in the extended zone. We can observe that the maximum size of zone at a level $i = 2$ is nearly 2.5% and 0.4% of the whole network for *Chandigarh* and *Hyderabad* respectively. We also observe that the maximum and average size of the extended zones drops with level i. This is because as the level increases more edges are added to the skeleton G_S, thus dividing the zones in a particular level i into smaller zones, thus leading to a drop in the maximum and average size of zones in the next level $i + 1$. The maximum and average size of a zone and size of skeleton are key factors in deciding the performance gain in path computation since the size of the reduced graph $G_{x,y}$ used for path computation comprises of the skeleton G_S and the extended zones($Z^*_{x^t}$, $Z^*_{y^t}$) where $x^t = z(v_x)$, $y^t = z(v_y)$. It should be noted that the skeleton at level 10 includes the entire road network which can be observed from Figure 4(a) and 5(a). This leads to an empty residual network and thus no zones are formed at level 10. Therefore the sizes of extended zones at level 10 are not shown in Figure 4(b) and 5(b).

6.1 Query Processing

Figure 4(c) and 5(c), illustrates the variation of average path computation time taken with the length of the path. On Y-axis is the path computation time in milliseconds. X-axis is numbered with the query set number q where $1 \leq q \leq 5$. For the sake of convenience the plots are shown for skeletons of level i where $i = 1, 2, 3, 4, 6$. The curve with the dashed line represents the computation time on the original graph $G(V, E)$. As the path length increases the computation time increases as more nodes and edges have to be processed in order to find the path. We can also observe that at level 2, the gain in path computation time achieved is at least 4-5 times not using more than 25 % of the total network for path computation.

Figure 4(d) and 5(d), illustrate the variation average path error with the length of the path. On Y-axis is the percentage error in path. X-axis is numbered with the query set number q where $1 \leq q \leq 5$. For the sake of convenience the plots are shown for skeletons of level i where $i = 1, 2, 3, 4, 6$. It can be seen that as the path length increases the path error gradually decreases. This indicates that the proposed model is more suitable for computing longer paths. We can also observe that, at level 2, for larger distances($q \geq 2$), path error is less than 2% while not using more than 25% of the total network.

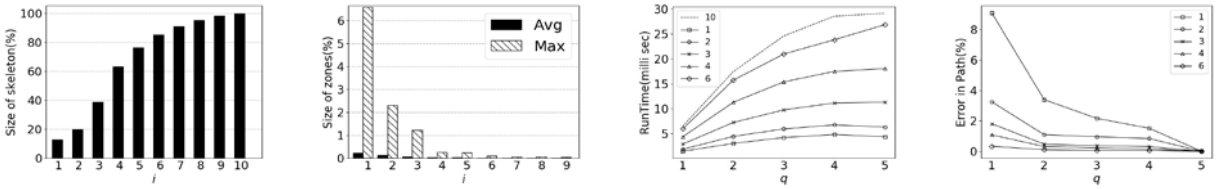
The optimal skeleton sizes and their respective computation gains and path errors are averaged and tabulated in Table 5 for all the 4 datasets given in Table 4.

Table 5. Performance Gain of Optimal Network Skeleton

Dataset	*Network(%)*	*Path Error(%)*	*Gain(X)*
Chandigarh	25	≤ 3	4.8
Hyderabad	26	≤ 2	4.5
NYC	24	≤ 5.5	5
Belgium	23	≤ 6	6.5

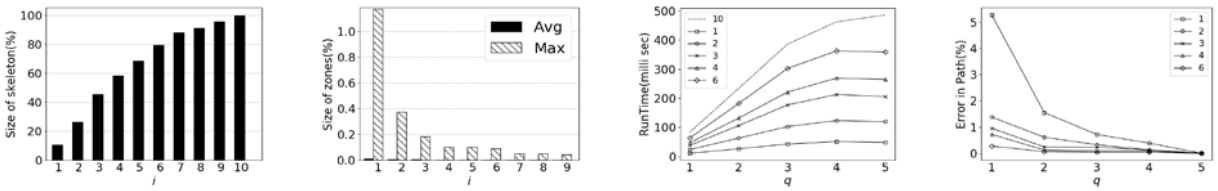
7. CONCLUSION

In this paper we present a *Skeletal Model* using map generalization technique to reduce the size of input network used for path computation. The proposed model is implemented in pgRouting to provide both improved path computation and structured way of storing and retrieving the network data used for path computation. A new path algorithm is proposed to compute path using the *Skeleton Model* by reusing the existing pgRouting Dijkstra



(a) Skeleton Sizes (b) Extended Zone Sizes (c) Computational Gain (d) Path Error

Figure 4. Chandigarh.



(a) Skeleton Sizes (b) Extended Zone Sizes (c) Computational Gain (d) Path Error

Figure 5. Hyderabad

algorithm. The implemented pgRouting path algorithm is tested on the real world road networks of varying size and topology. Results show that an average speedup of nearly 5X in path computation time was achieved while the processed network data was not more than 30% of the original road network. While these are significant gains, it has to be noted that these came at a cost of having an average path deviation error of less than 7%. The query time is faster since the *reduced* graph size is less which cuts down the overall search space for pgRouting path finding algorithms.

Finally, we hope that this work will aid in implementing navigational services on a low resource system too, thus leading to a paradigm shift from a traditional client-server model to the development of a client based path computation model. With the advancement in computing power and storage capacity of hand held devices, the focus should move towards utilizing these devices for path computation while reducing the dependency on network coverage or server response.

REFERENCES

Akimov, A., Kolesnikov, A. and Franti, P., 2004. Reference line approach for vector data compression. In: *Image Processing, 2004. ICIP'04. 2004 International Conference on*, Vol. 3, IEEE, pp. 1891–1894.

Bjørke, J. T. and Isaksen, E., 2005. Map generalization of road networks: Case study from norwegian small scale maps. In: *Proceedings XXII International cartographic Conference.*

Chondrogiannis, T. and Gamper, J., 2016. Pardisp: A partition-based framework for distance and shortest path queries on road networks. In: *Mobile Data Management (MDM), 2016 17th IEEE International Conference on*, Vol. 1, IEEE, pp. 242–251.

Geisberger, R., Sanders, P., Schultes, D. and Delling, D., 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: *International Workshop on Experimental and Efficient Algorithms*, Springer, pp. 319–333.

Jiang, B. and Claramunt, C., 2004. A structural approach to the model generalization of an urban street network. *GeoInformatica* 8(2), pp. 157–171.

Jiang, B. and Harrie, L., 2004. Selection of streets from a network using self-organizing maps. *Transactions in GIS* 8(3), pp. 335–350.

Jung, S. and Pramanik, S., 1996. Hiti graph model of topographical road maps in navigation systems. In: *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, IEEE, pp. 76–84.

Khoshgozaran, A., Khodaei, A., Sharifzadeh, M. and Shahabi, C., 2008. A hybrid aggregation and compression technique for road network databases. *Knowledge and Information Systems* 17(3), pp. 265–286.

Mackaness, W. A. and Beard, K. M., 1993. Use of graph theory to support map generalization. *Cartography and Geographic Information Systems* 20(4), pp. 210–221.

Möhring, R. H., Schilling, H., Schütz, B., Wagner, D. and Willhalm, T., 2005. Partitioning graphs to speed up dijkstras algorithm. In: *International Workshop on Experimental and Efficient Algorithms*, Springer, pp. 189–202.

Shekhar, S., Huang, Y., Djugash, J. and Zhou, C., 2002. Vector map compression: a clustering approach. In: *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, ACM, pp. 74–80.

Suh, J., Jung, S., Pfeifle, M., Vo, K. T., Oswald, M. and Reinelt, G., 2007. Compression of digital road networks. In: *International Symposium on Spatial and Temporal Databases*, Springer, pp. 423–440.

Thomson, R. C. and Richardson, D. E., 1995. A graph theory approach to road network generalisation. In: *Proceeding of the 17th international cartographic conference*, pp. 1871–1880.

Thomson, R. C. and Richardson, D. E., 1999. The good continuationprinciple of perceptual organization applied to the generalization of road networks.

Zhang, Z., 2006. Vector road network compression: a prediction approach. In: *Proceedings of the American Society for Photogrammetry and Remote Sensing Conference, ASPRS, Reno, Nevada, USA.*