UNIVERSITY OF CALIFORNIA
RIVERSIDE

Methodology Comparisons in Progressive Transfer of Vector-Based Geographic Data

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Ali Mirzabeigi

August 2011

Thesis Committee:
Vassilis J. Tsotras, Chairperson
Michalis Faloutsos
Eamonn Keogh

ABSTRACT OF THE DISSERTATION

Methodology Comparisons in Progressive Transfer of
Vector-Based Geographic Data

by

Ali Mirzabeigi

Master of Science, Graduate Program in Computer Science
University of California, Riverside, August 2011
Dr. Vassilis J. Tsotras, Chairperson

This paper presents how a single dataset containing a non-redundant representation of a vector map with highest level of details can be used to progressively refine current view of thin clients in client/server architectures. We built an index using a slightly modified version of the topological Generalized Area Partition (tGAP) to perform complete selection of edges required to build all face geometries for any arbitrary map scale. Some additional attributes are used in respective data models. Edge lengths are used in order to faster identify merging face objects during construction of the Face Tree structure. In addition, a geometric attribute for unified face MBRs is maintained to achieve the highest selectivity of edges in our spatial database queries.

Stateful clients with asynchronous communication approach have not been subject of prior works. We propose multi-threaded client and server designs that result in high performance applications. The client components track the result sets submitted from server and maintain a local structure every time they receive a response from the server. Server uses a lightweight RESTful web service with JSON serialization of the results to minimize network overhead besides minimizing the amount of data transferred over the media.

Multi-threading and asynchronous communications allow cancellation of an ongoing process in the server for previous requests of the clients. This feature efficiently reduces the overheads in both clients and server when users modify their area of interests by panning or zooming to a different map scale.

In our first implementation clients use simplex communication by asynchronously polling the data from server. Also, a duplex communication presented where server pushes the results to clients.

Three different methods for data retrieval from the spatial database proposed and evaluations performed. We show how spatial indexes as well as attribute indexes can hugely help better data retrievals in a fraction of a second. In addition, two alternatives to retrieve the data from RDBMS are described and we show how data retrieval with a forward-only stream of data rows can increase the performance even more.

Table of Contents

1. Introduction ---------- 1

2. Topological Schemes ---------- 4

2.1. The Left-Right Topology ---------- 5

3. The tGAP Index ---------- 7

3.1. The tGAP Face Tree ---------- 9

3.2. The tGAP Edge Forest ---------- 12

4. The Lean tGAP Index ---------- 13

5. The Constrained tGAP Index ---------- 14

6. Implementation and Experiments ---------- 15

6.1. Data Models ---------- 15

6.1.1. The Left-Right Topology Data Model ---------- 16

6.1.2. The tGAP Index Data Model ---------- 17

6.2. Datasets ---------- 18

6.3. Development Platform/Technology ---------- 18

6.4. The tGAP Index Construction ---------- 18

6.4.1. Building the tGAP Face Tree ---------- 19

6.4.1.1. Notes about the Algorithm ---------- 20

6.4.2. Building the tGAP Edge Forest ---------- 20

6.4.2.1. Notes about the Algorithm -- 21

6.4.3. Analysis on tGAP Face Tree and tGAP Edge Forest Algorithms --------------------- 24

6.4.4. Shrinking the tGAP Edge Forest Structure --- 25

6.5. System Architecture -- 26

6.5.1. XML vs. JSON -- 29

6.5.2. Asynchronous Communications -- 31

6.5.3. Maintaining the State -- 31

6.5.4. Polling Mechanism (Simplex Communication)--- 32

6.5.5. Pushing Mechanism (Duplex Communication)-- 32

7. Progressive Transfer of Vector Data -- 34

7.1. Database Queries -- 35

7.2. Effect of Database Indexes-- 38

8. Conclusion --- 42

9. References-- 44

List of Figures

Figure 1: Connectivity information in Arc-Node Vector Model.-------------------------------------- 6

Figure 2: Contiguity information in Arc-Node Vector Model. -- 6

Figure 3: Area Definition in Arc-Node Vector Model. -- 7

Figure 4: Area compatibilities and length of the shared boundary play an important role in the Collapse function.-- 9

Figure 5: The largest scale source map used for generalization of Face objects. Each polygon is associated with an importance value using the application-specific Importance function. --------10

Figure 6: Sequences involved in the Face merging process. --10

Figure 6, continued: Sequences involved in the Face merging process.-----------------------------11

Figure 7: The computed tGAP Face Tree for the map shown in Figure 5.--------------------------12

Figure 8: The tGAP Edge Forest as part of the tGAP Face Tree construction in Figure 6.--------13

Figure 9: The tGAP Edge Forest of Figure 5 after shrinking process.-------------------------------14

Figure 10: Entity relationship diagram in the Left-Right Topology scheme.------------------------16

Figure 11: The tGAP index entity relationship diagram. --17

Figure 12: Examples of edge processing during the face merge process. Edge $e7$ collapses when face $A4$ is merged by face $A1$ and its lifetime will be ended in the tGAP Edge Forest structure. 23

Figure 13: The worst case scenario in creation of the Edge Forest.----------------------------------25

Figure 14: The client/server architecture.---28

Figure 15: Amount of serialized data submitted from server in XML and JSON formats. --------30

Figure 16: Amount of serialized data submitted from server in XML and JSON formats. -------- 30

Figure 17: Progressive transfer of San Diego ZIP codes from server to client ---------------------- 34

Figure 18: Time needed to retrieve objects with each query option----------------------------------- 38

Figure 19: Actual RDBMS execution plan for queries in option I. ----------------------------------- 38

Figure 20: Actual RDBMS execution plan for queries in option II. ---------------------------------- 39

Figure 21: Actual RDBMS execution plan for queries in option III. --------------------------------- 39

Figure 22: The Effect of spatial and attribute indexes in option I. ------------------------------------ 41

Figure 23: The Effect of spatial and attribute indexes in option II. ----------------------------------- 41

Figure 24: The Effect of spatial and attribute indexes in option III. ---------------------------------- 42

List of Tables

Table 1: Information about the datasets used in experiments. -- 18

Table 2: Number of rows in Left-Right Topology and the tGAP Index. ---------------------------- 23

Table 3: tGAP index after applying the Edge Forest shrinking algorithm.--------------------------- 26

Table 4: A brief comparison between SOAP and REST protocols.---------------------------------- 27

Listings

Listing 1: The algorithm to build the tGAP Face Tree structure. -------------------------------------- 19

Listing 2: The algorithm for building the tGAP Edge Forest structure. ------------------------------ 21

Listing 3: SQL query to retrieve all the edges of San Diego ZIP codes with option I. ------------- 36

Listing 4: SQL query to retrieve all the edges of San Diego ZIP codes with option II. ------------ 37

Listing 5: SQL query to retrieve all the edges of San Diego ZIP codes with option III. ----------- 37

Methodology Comparisons in Progressive Transfer of Vector-Based Geographic Data

Ali Mirzabeigi

University of California, Riverside

August 2011

1. Introduction

Map generalization and progressive transfer of geographic data has been the subject of many researches in GIS and computer science for a long time. Researchers proposed various methods to avoid storage of redundant data and optimize communications between endpoints in client/server environments. Panning the map to a different extent or requesting for a new level of detail after zooming in or out are two major use cases in all mapping applications. Optimal visualization of the spatial data (map rendering) on the client-side requires these systems to minimize the amount of data being transferred over the network therefore; server-side generalization of data plays an important role to reduce overall response time of the entire system. Progressive transfer of geometries, on the other hand helps rendering process perform relatively faster on the client resulting more interactive applications as each time small amount of data rendered on the client. Smart thin/thick clients can reuse the data that have been already submitted from the server during the refinement steps in order to minimize both network traffic and the amount of data being transferred over the media.

For raster data this could be achieved by presenting a coarse quality of the image that could be drawn quickly first, then refining the view while the time is elapsing and user stays in the same map resolution. Raster data pyramids [18], and wavelet-based [19, 20, 21] methodologies can be used to achieve progressive transfer of raster images. JPEG2000 is an example of the data compression with wavelet approach.

For vector-based geographic data one can create several datasets for each predefined scale of an existing map and display the most appropriate/compatible scale when users request the map for a certain scale. This is so called Multiple Representation Data Bases (MRDB). Since this approach uses fixed number of predefined map resolutions there is no possible way of rendering data for every possible scale as only the closest resolution will be chosen and drawn. There is also the problem of maintaining redundant geographic data between the scales in which requires complex update processes besides the need for more memory to store the data. Another drawback is that MRDB cannot be used to progressively transfer the map objects as each resolution requires corresponding graphical representation of that scale transferred between clients and the server. In contrast variable scale approaches create and consume an index structure on the main dataset which is the largest scale of the map containing the most level of detail (LoD) for geographic objects and extract the requested scale on the fly. This server side generalization approach takes place before submitting the information to each client and will eliminate the need to create and store redundant datasets on the server. In addition, objects can be displayed at any arbitrary scale with faster response time experienced at each client.

There are several data structures available for the latter approach, but related works have proven that the topological Generalized Area Partition (tGAP) index is the most optimal generalization solution. tGAP can be used to create and consume several hierarchical indexes to progressively transfer vector-based geographic data in client/server environments. It uses offline generalization algorithms using spatial relationships between features to build hierarchical structures of faces and edges. A range of importance values are assigned to each object to identify which level of detail a face or edge geometry is valid. The upper bound values indicate that the object should not be transferred for map resolutions greater than or equal to that scale. Once the server receives a request for a desired level of detail, tGAP uses these importance ranges to retrieve respective features on the fly and clients would be able to receive and construct the complete set of geometries for that scale.

There were many research activities for tGAP index structures and different variations of them have been proposed. Nonetheless, the problem of how to efficiently communicate and transfer spatial data between the two endpoints in practical client/server architectures especially in the context of web and mobile applications that have limited network bandwidth has not been main focus of previous works.

This paper performs an analysis on different implementations of these index structures and presents how a slightly modified version of the tGAP with Left-Right Topology can be used to achieve progressive transfer of spatial data while minimizing both usage of network bandwidth and the amount of data stored on server which in turn increases overall system performance by minimizing execution times of related spatial queries.

We propose two different architectures and perform methodology comparisons. First, we design a client that progressively polls vector data from the server. The client application initiates the requests based on its current map extent and keeps track of the features that are drawn on its map view. Every time user interacts with the map to pan or zoom to a smaller/larger scale this multi-threaded application asks for termination of the previous request asynchronously (if it is still in progress) and polls the data for the new resolution. The advantage of multi-threaded architecture in the client is that the UI thread never gets blocked while the background thread is submitting the request and retrieving the spatial data from the server. In addition, by taking advantage of the tGAP data structures system allows the server to transfer spatial data from higher importance levels down to the lower levels based upon the hierarchical information that has been stored for all faces and edges, i.e. generalized versions of the faces will be transferred first and the client will progressively update the content of its map view by receiving lower important neighboring objects to obtain more details.

In our second design the client submits a request for the geographical data in a given map extent and server fetches the data from RDBMS. The difference with the previous approach is that this time server pushes data to clients and keeps track of the objects submitted to each client (session). A thread in each client retrieves objects from the server in the same fashion, i.e. objects with higher importance levels come and drawn first etc… Multi-threaded clients make it possible for users to interact with their map view while data are being pushed from the server and no blockage occurs in their UI thread.

Finally, we analyze and compare both methodologies and investigate the effect of database spatial indexes in their overall performance and show how both approaches can benefit from them.

2. Topological Schemes

Vector data can be stored either with their geometric or topological information. Geometric information contains GPS coordinates for all features and because of this all shared nodes and shared edges in face objects are stored multiple times (redundant data). In contrast topological information uses references to unique identifier of each node and edge object to eliminate this redundancy.

GIS applications use topology information to perform spatial analysis. Some examples are determining boundaries, containment, intersection, overlapping, left/right adjacency, etc… Topology refers to spatial relationships between geometry objects and it is necessary to perform any spatial analysis with vector data (points, lines/polylines, and polygons). Spatial analysis is computationally time consuming therefore; Topological schemes are designed to minimize the running time of such operations. They include topological structures to convert computational geometry algorithms into combinatorial algorithms. Non-Topological schemes on the other hand do not contain these complex structures and become more lightweight datasets for presentation purposes however; the lack of spatial relationships makes them computationally expensive to perform any spatial analysis.

This paper uses a topological scheme with Left-Right Topology for generalization of vector data. Respective objects can be stored in relational database management systems (RDBMS) with Node-Arc-Area structures. The next section briefly describes the Left-Right Topology definition.

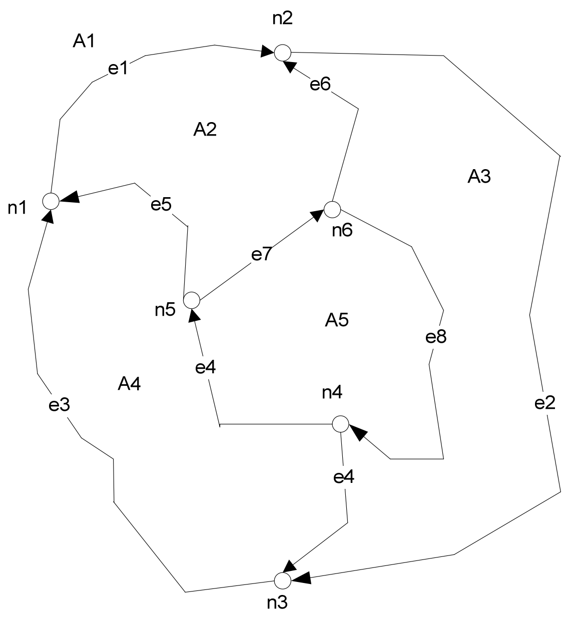
2.1. The Left-Right Topology

Left-Right Topology uses the Arc-Node vector model to store the spatial data. This model consists of a collection of node, arc, and area objects. Nodes are basically point features with their X and Y coordinates and are the basic units of the Arc-Node vector model.

For each arc, a list of references to its left and right areas is stored (Contiguity information). To construct a face, respective edges can be queried and joined to each other for example in Figure 2 we can build the geometry of face A_1 (referred as universe area) by using edges e_1, e_2, and e_3.

Arc objects are sets of line segments that connect two or more nodes. Respective nodes are linked in a particular order that is, all arcs have a direction. Arcs are defined by their start and end nodes (Connectivity information). To determine whether a set of arcs is connected common from and to nodes should be calculated. For example, in Figure 1 arcs e_6, e_7, and e_8 intersect because they all share node n_6.

An area is a series of arcs that form a closed boundary. In Arc-Node vector model an ordered list of arcs is used to represent an area object (Area Definition, Figure 3) therefore; arc coordinates are stored without coordinate redundancies ensuring boundaries of all adjacent polygons to not overlap with each other.



Connectivity List

Arc	From Node	To Node
e1	n1	n2
e2	n2	n3
e3	n3	n1
e4	n4	n3
e5	n5	n1
e6	n6	n2
e7	n5	n6
e8	n6	n4
e9	n4	n5

Figure 1: Connectivity information in Arc-Node Vector Model.

Contiguity List

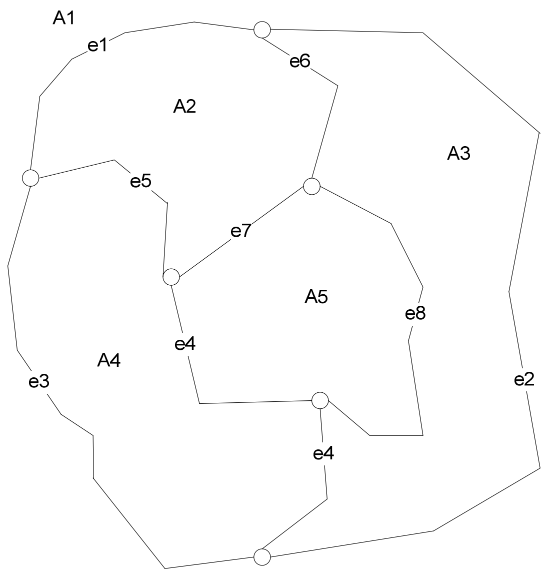
Arc	Left Area	Right Area
e1	A1	A2
e2	A1	A3
e3	A1	A4
e4	A3	A4
e5	A4	A2
e6	A2	A3
e7	A2	A5
e8	A3	A5
e9	A4	A5

Pseudo SQL Query:

```
Select [Arc] FROM [Contiguity List]
WHERE [Left Area] = A1 OR [Right Area] = A1
```

{ e1, e2, e3 }

Figure 2: Contiguity information in Arc-Node Vector Model.



Area Definition

Area	Arc List
A1	{ e1, e2, e3 }
A2	{ e1, e6, e7, e5 }
A3	{ e2, e4, e8, e6 }
A4	{ e3, e5, e9, e4 }
A5	{ e7, e8, e9 }

Figure 3: Area Definition in Arc-Node Vector Model.

3. The tGAP Index

The tGAP uses offline generalization algorithms to build hierarchical structures of Face and Edge objects by consuming the Left-Right Topology scheme of the largest scale map in which contains representations of the entire objects with the most details. These generalization processes transform the map with highest level of detail to a set of smaller scales with objects referencing to the original dataset.

There are two major problems related to partitioning of the Area objects. First, collapsing an Area object will produce a hole which is not acceptable in map cartography and second, generalization of an Arc object in common boundaries of Area objects may cause overlapping features or a gap between the Area objects. The latter can be addressed by using the Left-Right Topology scheme. For the former problem tGAP uses neighboring features to cover the hole introduced by leaving out an Area. This could be done as part of the generalization process that runs in multiple steps. At each step the least important Area is collapsed to its most compatible neighbor therefore; an Importance function is needed to decide which area should be removed at each step plus, how the gap introduced by this removal should be filled. A generic Importance function for Area A can be defined as the following:

$$Importacne(A) = f(Area(A), WeightFactor(A))$$

The $WeightFactor$ function depends on category of the Area object hence tights to purpose of the map, and its definition varies from one application to another. For example, one application may define a building block more important than a grass land area.

tGAP uses the Collapse function to perform a merge process. For two neighboring Areas A_1 and A_2 that are selected by the Importance function in the previous step the Collapse function is defined as:

$$Collapse(A_1, A_2) = g(BoundaryLength(A_1, A_2), Compatibility(A_1, A_2))$$

Besides the length of the shared boundary a $Compatibility$ function is used to provide a scoring mechanism to determine how the two Areas A_1 and A_2 that might belong to different classes or categories can be merged together. The Pond area in Figure 4 has two candidate neighbors Park and Town for Collapse operation. Since Park and Pond have a higher compatibility score (defined in the application) and length of the shared boundary between them is longer than the boundary between Pond and Town, the generalization process collapses the Pond into the Park feature and selects Park as the class/category of the newly generated Area.

These transformation steps of the geometries during the generalization process are stored in tGAP structures and can be used for progressive transfer of vector data between the endpoints.

All tGAP indexes are physically stored in a RDBMS using Node, Edge, and Face tables. Nodes include the start and end points of each Edge object. Edges form a forest of polylines that contain references to their left and right Faces and finally the Face table includes a tree of Area objects in the topological scheme. The hierarchical structures of Edge and Face tables use importance range values to specify when an instance of corresponding objects is valid and should be drawn in a given scale.

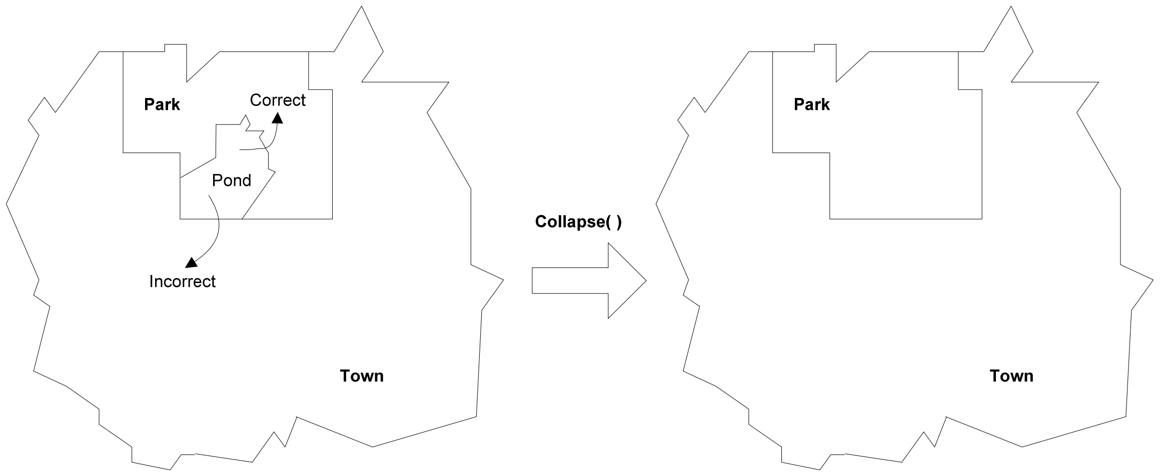


Figure 4: Area compatibilities and length of the shared boundary play an important role in the Collapse function.

The next sections describe the algorithms that create these Edge and Face hierarchies in tGAP structures.

3.1. The tGAP Face Tree

The algorithm applies the Importance function described above to a collection of Face objects in the large scale dataset. At each step the Face with the lowest importance value is removed and the merging process uses the application-specific Collapse function to create a new object with regards to feature class/category of its most compatible neighbor and sets the parent of both objects to the new Face. In addition, an importance value will be computed for the inserted Face as the summation of merged objects importance values.

This process will continue until one huge object (the Universal Face) is left. Figure 6 shows the sequence of steps to build the tGAP Face tree of the map in Figure 5. The Universal Face (A_{11}) forms the root of the tGAP Face tree index.

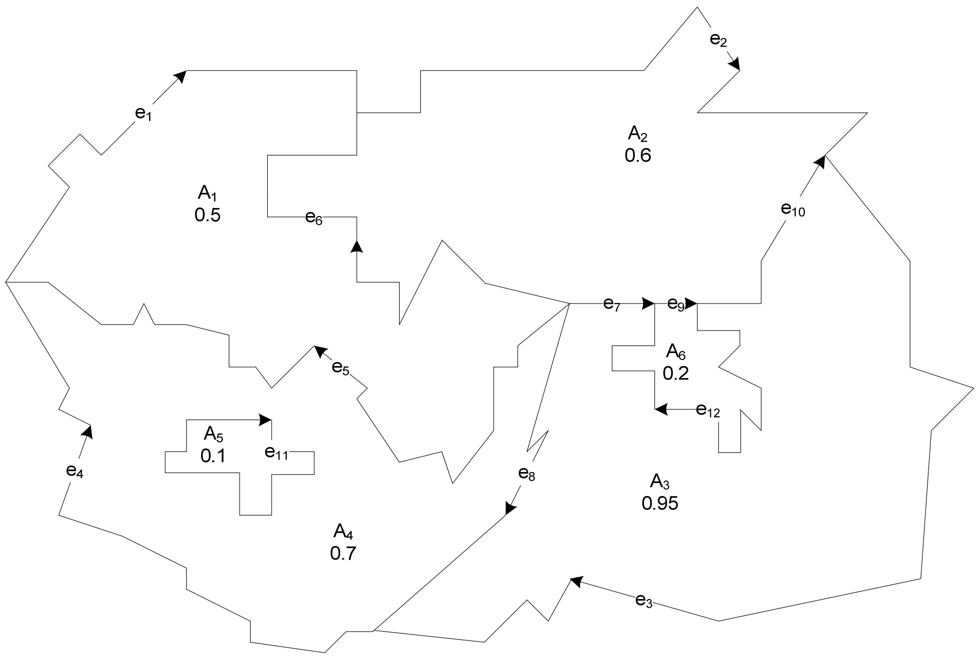


Figure 5: The largest scale source map used for generalization of Face objects. Each polygon is associated with an importance value using the application-specific Importance function.

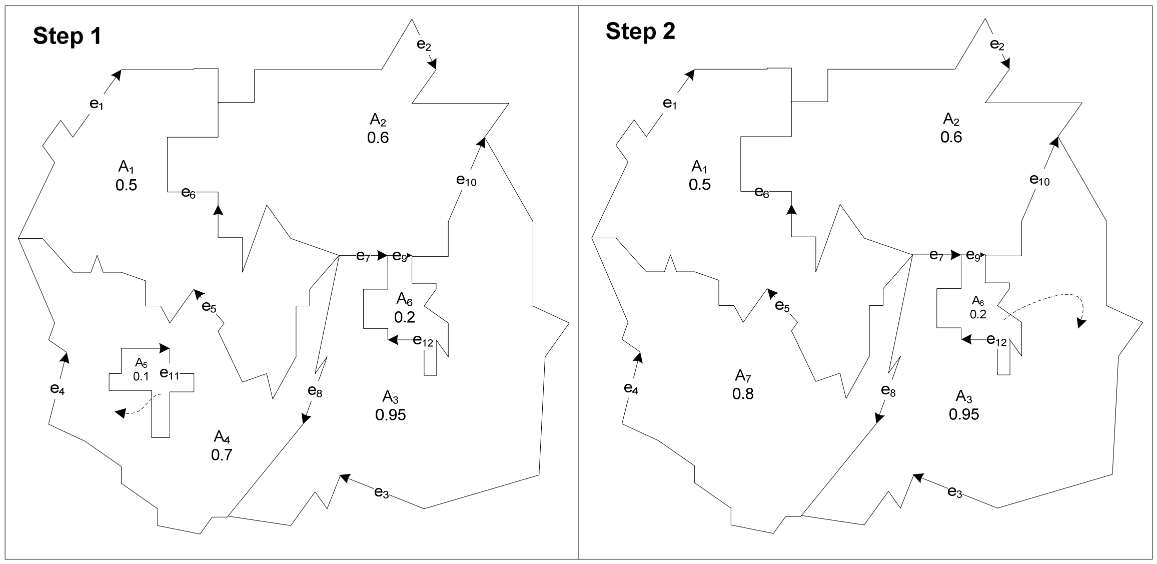


Figure 6: Sequences involved in the Face merging process.

The merge process also associates lower and upper bound values of the importance ranges to each Face object. These values specify lifespan of each object and allow selection of appropriate Faces for a requested LoD. The lower bound importance values of all the leaf nodes in the Face tree are assigned to zero to identify objects with the highest level of details. An importance level will be computed at each step of the merging process and assigned as the upper bound importance values of the merged objects. The new parent Face uses this value as its lower bound importance range. Figure 7 shows the computed tree for the large scale map shown in Figure 5.

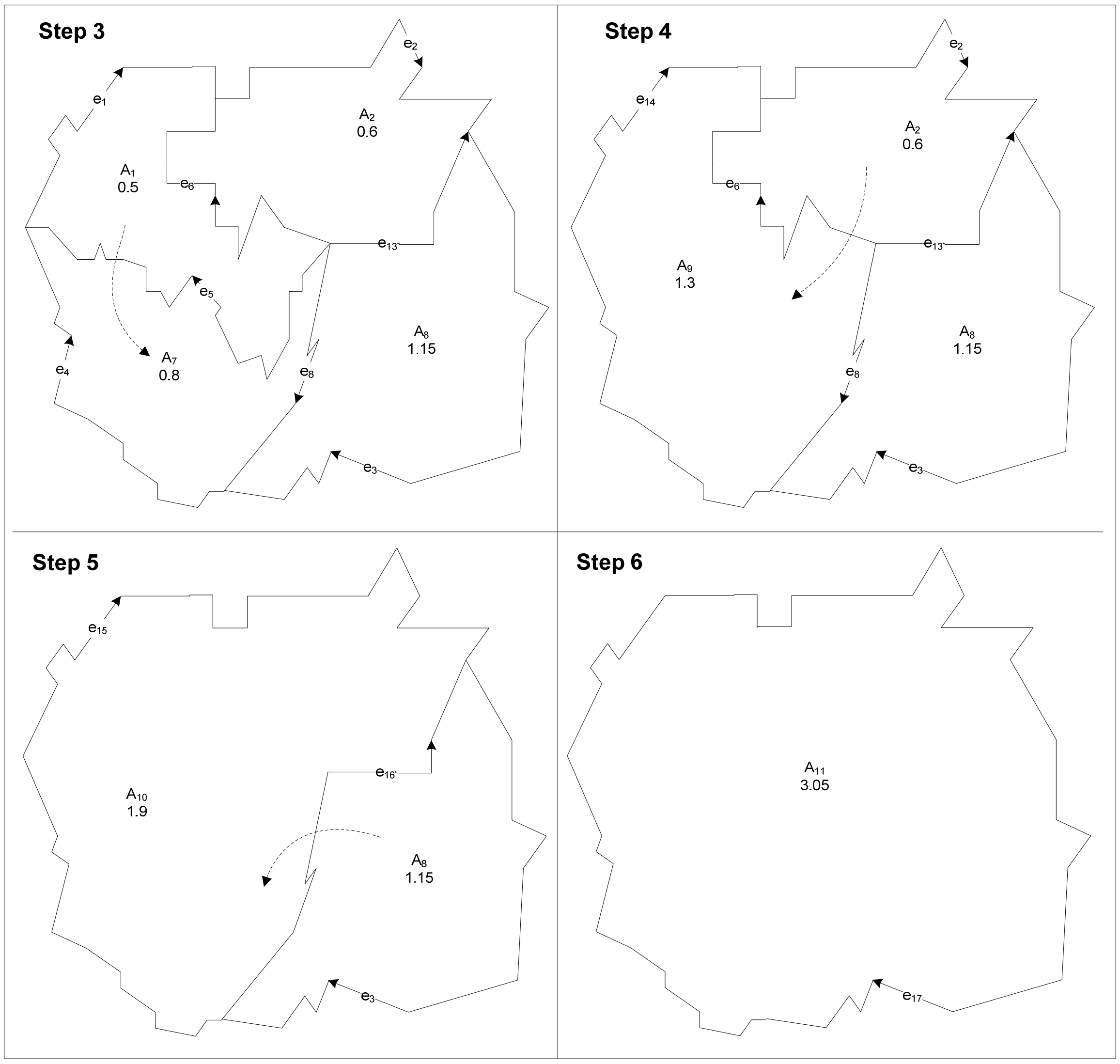


Figure 6, continued: Sequences involved in the Face merging process.

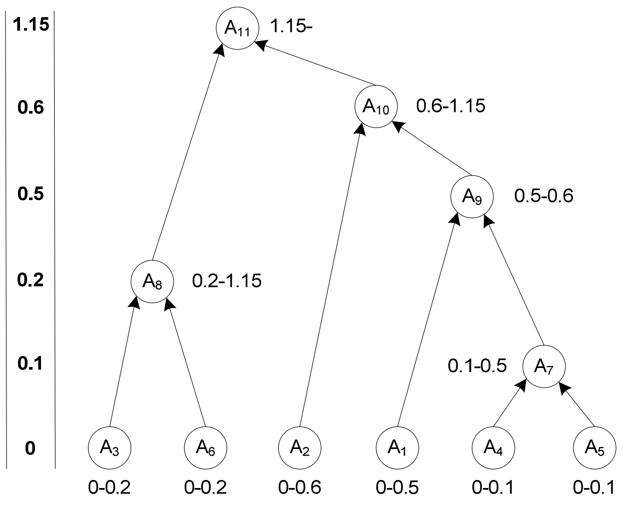


Figure 7: The computed tGAP Face Tree for the map shown in Figure 5.

3.2. The tGAP Edge Forest

During construction of the tGAP Face tree the lifetime of edges might be changed with one of the following three scenarios:

I. Removal of an Edge in the common boundary of two Faces.

In the sample shown in Figure 6 e_{11} in step 1 is completely removed to build the Face A_7.

II. Change in left or right Face references of an Edge.

During second step of the Face merging process in Figure 6 the left Face reference for e_5 and right Face references for Edges e_4 and e_8 have changed from A_4 to A_7.

III. Merging of two or more Edges to build a new Edge.

An example of this case is the merge of Faces A_6 and A_3 in step 2 in which Edges e_7, e_9 and e_{10} have joined to build e_{13}.

The tGAP Edge Forest is constructed by a set of original Edges, collapsing Edges, versioned Edges, and merged Edges. Original Edges are stored as leaf nodes and they contain respective geometry coordinates. For collapsing Edges (scenario I) and versioned edges (scenario II) a reference to respective original Edge will be maintained as they have the same geometry definitions with the only difference that for the latter updated references to their left or right Face objects will be kept. Merged edges (scenario III) are constructed by sticking related edges to each other and stored by their corresponding geometries. Like Face objects in the tGAP Face tree structure each Edge is assigned an importance range to perform selection of appropriate Edge objects for a specific level of detail. As shown in Figure 8 different types of each Edge object maintained in the structure. Since Edge removals create local root nodes (shown as underlined bold nodes) there will not be a single root node in these hierarchies hence; a forest of Edges is created in the structure.

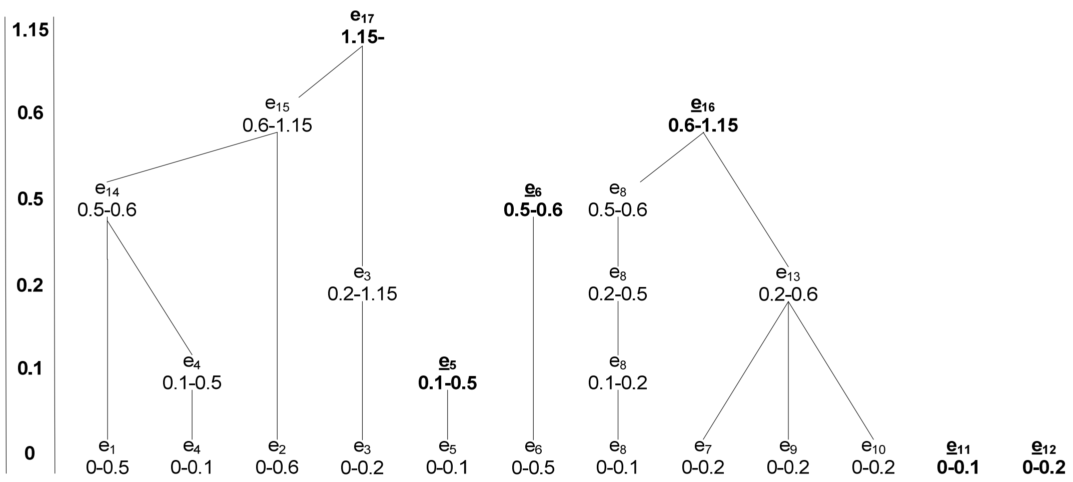


Figure 8: The tGAP Edge Forest as part of the tGAP Face Tree construction in Figure 6.

4. The Lean tGAP Index

During construction of the Edge Forest multiple references to edges are stored as part of the Edge versioning process (scenario II in section 3.2). Since there is no change in Edge geometries in these cases one may consider merging such entries to preserve more storage space as they are referencing to the same instance. The problems are how the lower and upper bound values of the importance ranges should be assigned to the aggregated object and how should we deal with the changes in the left and right Face references in each version.

Meijers et.al [6] proposed a shrinking algorithm that applies to the tGAP Edge Forest structure. They resolved the above problems by unification of all importance values which are always adjacent ranges and used the left Face reference on the lower bound and the right Face reference on the upper bound of the unionized importance ranges. In Figure 8, Edge e_8 has four versions due to the changes on its left or right Face references and the union of importance ranges for this edge will be:

$$(0, 0.1) \cup (0.1, 0.2) \cup (0.2, 0.5) \cup (0.5, 0.6) = (0, 0.6)$$

Therefore; the new aggregated versioned Edge will have A_3 as its left and A_8 as its right Face references.

Figure 9 shows the result of shrinking process on the Edge Forest structure shown in Figure 8.

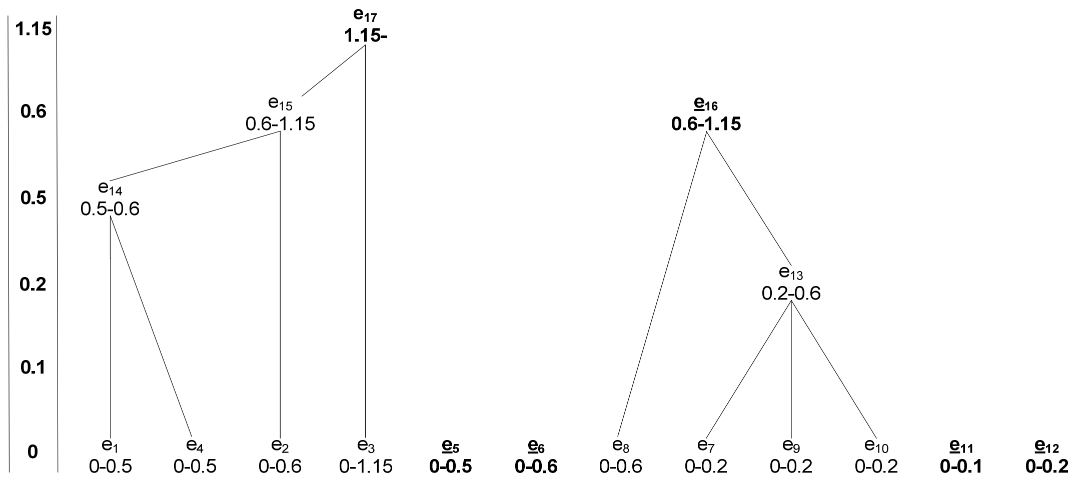


Figure 9: The tGAP Edge Forest of Figure 5 after shrinking process.

5. The Constrained tGAP Index

Some mapping applications use a large scale and a smaller scale map representations where the latter is produced from an independent process, i.e. the small scale map cannot be derived directly from the large scale map. The Constrained tGAP structure uses the two datasets and transforms the map with the highest level of detail into the one with smaller resolution.

Since the two datasets are available region constraints will be associated to each Face object. The Constrained tGAP [3, 4] index uses these composite regions during its object merging processes. The composite regions are provided with regards to application and cartographical needs and are available in the dataset. The hierarchical structures are stored by applying the same generalization steps on compatible Face and Edge objects using the same approach of the Importance function however; the Collapse function only allows the objects that belong to the same region constraint to be merged with each other.

The algorithm works as the following. Face objects with the lowest importance values are picked at each step and the most compatible neighbor within a compatible region is used for the Face merging process. The tGAP Edge Forest will be updated with collapsed, versioned, or merged Edge objects at each step of the Face merging computations. The calculations stop when the universal face is generated.

6. Implementation and Experiments

In this section first we describe the data models and algorithms used to store and populate each component of the tGAP index structure. We perform an analysis on memory requirement for the standard and lean tGAP indexes and show the implementation of our stateful client architectures. We show how database queries should be develop to fetch edge geometries with their importance and finally perform an analysis on each option.

6.1. Data Models

In order to maintain result of the offline generalization processes we need a set of tables to store respective objects. All tGAP structures are stored in a RDBMS. In this section descriptions of relationships between all entities that are used in construction of the tGAP index structure are provided.

6.1.1. The Left-Right Topology Data Model

To build the Left-Right Topology schemes ArcObjetcs programming with ESRI ArcCatalog extensions is used for corresponding Shape files of each input dataset (6.3). The result of this process is stored in our RDBMS. All geometry objects are validated using the RDBMS validation function. The $MakeValid()$ function in Microsoft SQL Server 2008 R2 guarantees all geometry instances on each table is in compliance with the rules specified in Open Geospatial Consortium (OGC). To perform this operation effectively we used $STIsValid()$ method on geometry instances of all rows to get features that do not have valid geometries, then for each row in the result set updated the coordinates with its valid definition. Figure 10 shows Left-Right Topology data model of our system.

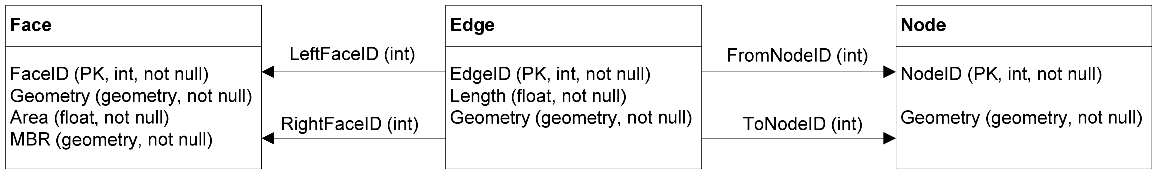

Face
FaceID (PK, int, not null)
Geometry (geometry, not null)
Area (float, not null)
MBR (geometry, not null)
LeftFaceID (int)
RightFaceID (int)
Edge
EdgeID (PK, int, not null)
Length (float, not null)
Geometry (geometry, not null)
FromNodeID (int)
ToNodeID (int)
Node
NodeID (PK, int, not null)
Geometry (geometry, not null)

Figure 10: Entity relationship diagram in the Left-Right Topology scheme.

During this operation we also calculated the Minimum Bounding Rectangle (MBR) of each Face object using the $STEnvelope()$ method for geometries of all Face objects. This column is stored in related table of the topological model and is used during construction of the tGAP Face Tree structure to calculate aggregated MBR of the merging faces. A geometry attribute will be used to store these columns in corresponding tables. For our experiments the column that stored in the tGAP Face Tree structure will be used in order to analyze the effect of spatial queries performed on the envelopes containing the Face objects versus the edge objects in the Edge Forest structure.

In addition, each Edge geometry length is also calculated and stored as an attribute in the Edge table. This helps us to find Faces that share the longest boundaries during construction of the tGAP Edge Forest structure with minimal calculations.

6.1.2. The tGAP Index Data Model

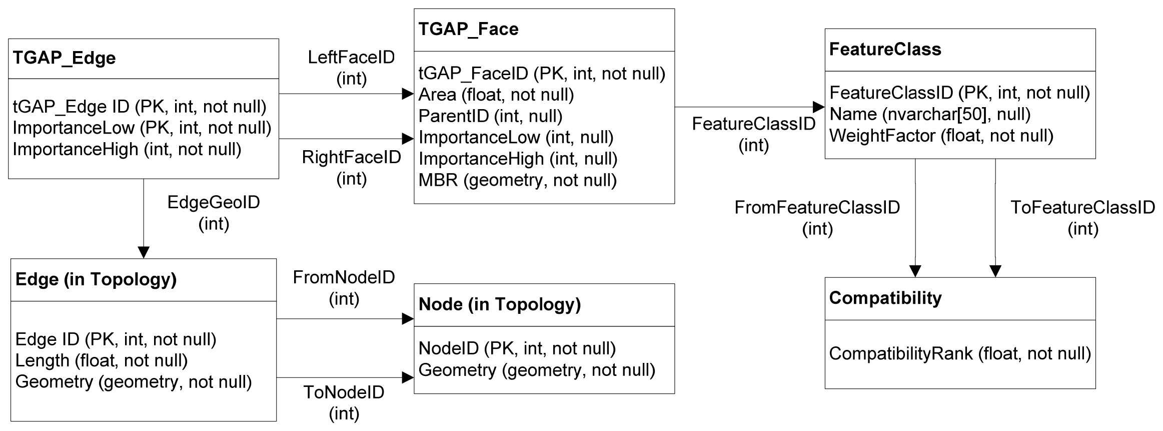

TGAP_Edge
tGAP_Edge ID (PK, int, not null)
ImportanceLow (PK, int, not null)
ImportanceHigh (int, not null)
LeftFaceID (int)
RightFaceID (int)
TGAP_Face
tGAP_FaceID (PK, int, not null)
Area (float, not null)
ParentID (int, null)
ImportanceLow (int, null)
ImportanceHigh (int, null)
MBR (geometry, not null)
FeatureClassID (int)
FeatureClass
FeatureClassID (PK, int, not null)
Name (nvarchar[50], null)
WeightFactor (float, not null)
EdgeGeoID (int)
FromFeatureClassID (int)
ToFeatureClassID (int)
Edge (in Topology)
Edge ID (PK, int, not null)
Length (float, not null)
Geometry (geometry, not null)
FromNodeID (int)
ToNodeID (int)
Node (in Topology)
NodeID (PK, int, not null)
Geometry (geometry, not null)
Compatibility
CompatibilityRank (float, not null)

Figure 11: The tGAP index entity relationship diagram.

During construction of the tGAP Edge Forest we showed that two types of Edge objects exist in the structure. Collapsing and versioned Edges (cases I and II in section 3.2) do not contain their geometries and they reference to respective Edge objects that are stored in the topological model. For merged Edges (case III in section 3.2) instead of adding new instances with their geometries in the database one may use Binary Line Generalization (BLG) Trees described in [11] to consume hierarchical references to corresponding Edge geometries, but this structure requires tracing a lot of references during construction of Face objects which in turn results an extra load on the state-full clients since they would need to keep track of many hierarchical references on their end as well as the need for joining too many Edges during the construction of Face objects which is not desired while drawing the objects on their maps. Therefore; it is decided to compromise the need to store these redundant merged Edge geometry representations on the server to gain a higher performance on client sides as well as minimizing their complexities.

6.2. Datasets

We used three different datasets from the U.S. Census Bureau [14]. Datasets include Census 2000 5-Digit ZIP Code Tabulation Areas (ZCTA) for Wyoming, California, and the United States. All datasets are available in shape files for each state and a merging process has taken place to create a full dataset for the entire United States.

Since the source datasets do not contain any information about the spatial reference identification (SRID) also a projection to WGS-1984 (4326) performed for coordinates of all geometric objects. Table 1 shows the amount of information available in each dataset.

Dataset	Size (MByte)	# Faces	# Edges	# Nodes
Wyoming	1.19	285	764	518
California	4.97	2,080	7,545	5,564
United States	98.8	37,068	116,295	85,373

Table 1: Information about the datasets used in experiments.

6.3. Development Platform/Technology

Microsoft SQL Server 2008 R2 Express Edition is used as database management system. Client and server components are developed by Microsoft .Net Framework 4.0 using C# (CSharp) programming language. The server component is hosted by Microsoft Internet Information Services (IIS) 6. We used ESRI ArcGIS Desktop for all spatial calculations needed during the data preparation phase. ESRI ArcGIS Silverlight/WPF/WP7 API 4 is used in clients for map drawing and rendering processes.

6.4. The tGAP Index Construction

Construction of the tGAP structures is quite expensive especially for large datasets and cannot be applied on the fly therefore; all generalization algorithms should be performed offline and stored in respective RDBMS tables. The following sections describe the algorithms used for building the related structures in tGAP index.

6.4.1. Building the tGAP Face Tree

Listing 1 shows the algorithm used to generate the tGAP Face Tree hierarchy. To construct the tGAP Face Tree a priority queue of Face objects with ascending order of their Importance values is used. Since all Faces belong to the same class/category (ZIP codes) every Face F_i in the datasets is associated with the same weight factor.

```
Algorithm Build_tGAP_Face_Tree( )
// Uses topological datasets for Face, Edge, and Node geometries to build the tGAP
// Face Tree structure.
// Output: the tGAP Face Tree structure.
1.  Initialize tGAP_Face table by inserting all rows from the Face table. Set all
    ImportanceLow values to zero (ImportanceHigh values remain NULL).
2.  Build a priority queue $Q$ of tGAP_Face objects with ascending order of their areas.
3.  Set a global CurrentImportance value to one.
4.  Dequeue the face with the lowest importance value ($F_1$).
5.  While $Q$ is not empty do:
6.          Find the neighboring face $F_2$ with the largest area that shares the longest
            boundary with $F_1$.
7.          Merge $F_1$ with $F_2$, and insert the new face $F_{new}$ with new tGAP_FaceID
            in tGAP_Face table.
8.          For the new face $F_{new}$ do the following:
            - Assign $Area(F_{new}) = Area(F_1) + Area(F_2)$
            - Assign $MBR(F_{new}) = MBR(F_1) \cup MBR(F_2)$
            - Set ImportanceLow value to the CurrentImportance value.
            - Leave ImportanceHigh value to be NULL and Enqueue it to $Q$.
9.          Set the ParentID of $F_1$ and $F_2$ to $F_{new}$.
10.         Call Build_tGAP_Edge_Forest ( ) to create new edge versions and/or
            merged edges.
11.         Remove faces $F_1$ and $F_2$ from $Q$ and set their ImportanceHigh values to
            the CurrentImportance value.
12.         Increment the global CurrentImportance variable by one.
13.         Dequeue the face with the lowest importance value.
14. End While
```

Listing 1: The algorithm to build the tGAP Face Tree structure.

For the same reason all Faces use the same CompatibilityRank in the Compatibility table (Figure 11). Therefore; $Area(F_i)$ is used as the Importance function during construction of the Face Tree structure.

6.4.1.1. Notes about the Algorithm

- Since we have stored the Edge lengths in the Edge table line 6 could be easily done by finding the Edge object with the biggest value of its Length attribute in which has F_1 as its left or right face.
- The merge operation of the two faces in line 7 is done by adding a new node in the tree and summation of areas of the two child face objects as the area of the parent face as well as using the $STUnion()$ method of geometries in SQL to calculate the MBR geometry for the new Face (line 8).
- By leaving out the ImportanceHigh value of F_{new} to NULL in line 8, the new Face will be considered during the next generalization steps till one huge Face (the Universal Face) is generated.

6.4.2. Building the tGAP Edge Forest

The algorithm to build the tGAP Edge Forest is shown in Listing 2. As described in section 3.2 during construction of the tGAP Face Tree, three conceptual cases may happen to the Edges involved in the process. A collapsing Edge should be added when its corresponding Edge is in the common boundary between the two Faces that are merging with each other. A versioned Edge should be added whenever there is a change in the left or the right Face references of the original Edge during the Face merging process. For the remaining Edges a composite Edge may also be created as a result of joining two or more Edges when Faces contain Edges with common start or end Nodes.

```
Algorithm Build_tGAP_Edge_Forest( )
// Uses topological datasets for Face, Edge, and Node geometries to build the tGAP
// Edge Forest structure.
// Inputs: $F_1$, $F_2$, $F_{new}$ and the Current importance value in Face merging process.
// Output: the tGAP Edge Forest structure.
1. Initialize tGAP_Edge table by inserting all rows from the Edge table. Set all
   ImportanceLow values to zero (ImportanceHigh values remain NULL).
2. Build a global list $L$ of edges from the Edge table.
3. Create a local set of edges $ESet$ by removing the Edges from $L$ that belong
   to any of the input faces $F_1$ and $F_2$.
4. For each Edge $E$ in $ESet$ do:
5.         If $E$ is the longest common boundary edge of $F_1$ and $F_2$ then:
           - Make $E$ a local root node by removing it from $ESet$ and setting
             its ImportanceHigh value to the CurrentImportance value.
6.         If the edge $E$ can still be used after the face merging process then:
           - Create a new edge $E_{new}$ by removing edge $E$ from $ESet$ and setting
             ImportanceLow value of $E_{new}$ to the CurrentImportance value.
           - Set left and right face references of $E_{new}$ accordingly. Face $F_{new}$
             will be one of the two references.
           - Insert $E_{new}$ into tGAP_Edge table with the same ID as $E$.
           - Add $E_{new}$ to the global list of edges $L$ to be used in the next steps.
           - Make $E$ a local root node by setting its ImportanceHigh value
             to the CurrentImportance value.
7.         If edge $E$ should be joined by its neighboring edges then:
           - Create a new edge $E_{new}$ by merging edge $E$ to all of its neighboring
             edges that are remaining in the local set of edges $ESet$.
           - Remove all remaining edges from $ESet$ and set their ImportanceHigh
             values to the CurrentImportance value.
           - Set left and right face references of $E_{new}$ accordingly. Face $F_{new}$
             will be one of the two references.
           - Insert $E_{new}$ into tGAP_Edge table with a new EdgeID.
           - Insert $E_{new}$ into Edge table with a new EdgeID and store its geometry
             coordinates. Also, calculate the length of this edge and store it in the
             Length column of the table.
8. End For
```

Listing 2: The algorithm for building the tGAP Edge Forest structure.

6.4.2.1. Notes about the Algorithm

- During construction of the tGAP Edge Forest structure the global list L of edges from the Edge table always contains instances of edges from tGAP_Edge table with ImportanceHigh value of NULL. The edges are removed from this list when their ImportanceHigh value is set to the value of CurrentImportance variable.

- The common boundary edge with the longest length collapses during the face merging process of the given input polygons (edge e_7 in Figure 12). By setting the ImportanceHigh value of that instance (line 5) a local root node will be created in the structure. There is no need to copy a new version of this edge as the lifetime of the edge has been ended by removing it from the local edge set.
- Versioned edges (line 6) are inserted into the tGAP_Edge table with the same ID as their respective edges in the Edge table as they can still be used after creation of the new face object F_{new}. These edges are the ones that belong to either one of the input face objects in such a way that for each edge E, all its intersecting edges in $ESet$ have references to the face objects on their other side that are different than what edge E has.

  Figure 12 shows an example of such edges. Edges e_5 and e_6 follow the second category since they belong to the merging (least important) face A_4 but have different faces (A_2 and A_3) as their left faces therefore; a new version of these edges should be generated.

  The other case is when only one of the references to neighboring faces of an edge changes during the face merging process. An example is shown in Figure 12. Edges e_1, e_4, and e_8 belong to the most important face A_1, and because they are needed to create the parent face $A_5 = A_4 \cup A_1$ (not shown in the picture) a copy of each (new versions) with left or right references to the new face A_5 should be inserted into the structure.
- If edge E in line 7 belongs to either one of the input face objects and all its neighbor edges in $ESet$ have the same neighboring face as E does then all the edges that satisfy this criterion will be joined to each other to form a new instance of the merged edge. This edge will be added to the tGAP_Edge table with a new ID and will also be stored in the Edge table to maintain its geometry coordinates. An example of such edges is e_7, e_9, and e_{10} during second

step of the face merging process in Figure 6. All these edges have A_2 as their left face and are merged to form the composite edge e_{13}.

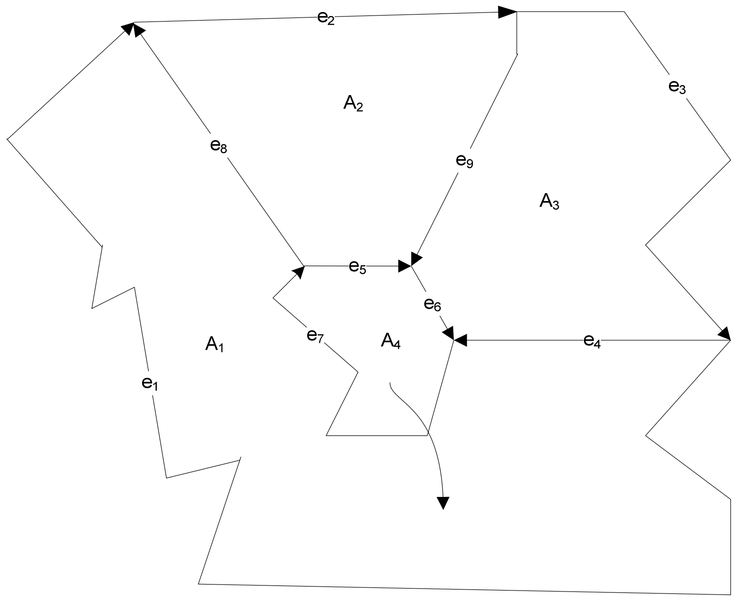


Figure 12: Examples of edge processing during the face merge process. Edge e_7 collapses when face A_4 is merged by face A_1 and its lifetime will be ended in the tGAP Edge Forest structure. Edges e_1, e_4, and e_8 will get new versions as their left or right face references are changed. Edges e_5 and e_6 also get new versions as they have different neighbors on their left side.

Table 2 shows size of the tGAP index after applying the above algorithms to our different input datasets.

Dataset	**Left-Right Topology**			**tGAP Index**		
	# Faces	**# Edges**	**# Nodes**	**# Faces (Inc. Factor)**	**# Edges (Inc. Factor)**	**# Nodes**
Wyoming	285	764	518	569 (~ 2x)	6,886 (~ 9.0x)	518
California	2,080	7,545	5,564	4,159 (~ 2x)	91,330 (~ 12.1x)	5,564
United States	37,068	116,295	85,373	74,135 (~ 2x)	1,663,018 (~ 14.3x)	85,373

Table 2: Number of rows in Left-Right Topology and the tGAP Index.

6.4.3. Analysis on tGAP Face Tree and tGAP Edge Forest Algorithms

Observation 1: Looking at Table 2 it can be observed that number of faces stored in the tGAP index is linear to number of faces in the source topology dataset, i.e. if the total number of faces in the topological dataset is equal to $|F|$, the tGAP index will require $O(|F|)$ number of faces in its structures.

Proof: The face merge process continues until one universe face is generated. For $|F|$ number of faces this generalization process requires $|F| - 1$ steps. Since each time a new face is added to the hierarchy the total number of faces in the structure is equal to $2|F| - 1$ which is $O(|F|)$.

Observation 2: Experiment shows that number of edges stored in the tGAP index could be quadratic to number of edges in the Left-Right topology dataset.

Proof: This is due to the fact that all adjacent faces share at least one edge, i.e. at least one edge should be removed during the face merging process. In the worst case scenario all edges may require new left and right references in their contiguity list therefore; the upper bound value for the number of edges stored in tGAP indexes is equal to $\sum_{i=0}^{|F|-1}(|E| - i)$ where $|E|$ is total number of edges.

Considering the extreme case shown in Figure 13 where all faces including the universe face contain only one edge the number faces would be equal to the number of edges in above relation, i.e. $|F| = |E|$.

Therefore; number of edges stored in the tGAP Edge Forest structure will be at most [6]:

$$\sum_{i=0}^{|F|-1}(|E| - i) = \sum_{i=0}^{|E|-1}(|E| - i) = |E|\left(\frac{|E| + 1}{2}\right) \cong O(|E|^2)$$

In practice no such extreme cases happen at all time and as shown in Table 2 the average number of edges stored in our tGAP index seems to be a factor less than 15 (average of ~12) of the edges in the topological dataset (between 9 and 15 in our test datasets).

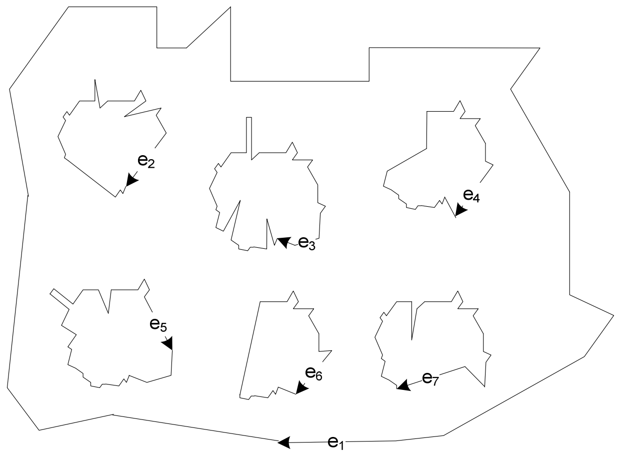


Figure 13: The worst case scenario in creation of the Edge Forest.

6.4.4. Shrinking the tGAP Edge Forest Structure

We also used the tGAP Edge Forest shrinking algorithm described in section 5. We unionized all entries that belong to different versions of the same edge. Different versions edges could be shrunk into one new edge entry by using the lower bound importance value of the first edge as well as its left face reference and the upper bound importance value of the last version with its right face reference. Corresponding rows as well as the intermediate versions have been removed from the tGAP_Edge table in the database. Table 3 shows the outcome of this process.

Observation: Number of edges stored in the tGAP Edge Forest structure after applying shrinking algorithm is $O(|E|)$.

Proof: All original edges are stored in the structure. This set grows with each merging process at the level of edge objects. The worst scenario is the case where all edges need to be merged. Using the same approach of face merging process two edges can be merged at each time until one huge polyline remains therefore; according to Observation 2 in section 6.4.3 the total number of edges will be $2|E| - 1$. On the other hand face merging requires removing at least one edge resulting

$|F| - 1$ edges being merged. Taking the two steps into account the total number of edges will be at most: $2|E| - 1 - (|F| - 1) = 2|E| - |F| \cong O(|E|)$

Dataset	tGAP Index			Lean tGAP Index		
	# Faces	**# Edges**	**# Nodes**	**# Faces**	**# Edges (Comp. Ratio)**	**# Nodes**
Wyoming	569	6,886	518	569	1,092 (~ 15.9)	518
California	4,159	91,330	5,564	4,159	11,642 (~ 12.7)	5,564
United States	74,135	1,663,018	85,373	74,135	198,926 (~ 12.0)	85,373

Table 3: tGAP index after applying the Edge Forest shrinking algorithm.

6.5. System Architecture

Figure 14 depicts both the client and the server architectures in the experiments. All HTTP requests submitted to a tGAP Web Service hosted in an IIS server. Two protocols can be used for exchanging our structured data through the network: Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) protocol. The former uses Extensible Markup Language (XML) as its output format. In contrast REST supports different types of formats. A lightweight fat-free alternative to XML, JavaScript Object Notation (JSON) is used for interchanging data in our RESTful Web service. More information about protocols and formats can be found in World Wide Web Consortium (W3C) and JSON websites [12, 13]. However; suitability of each format and protocol depends on the application requirements. Table 4 lists some of the pros and cons in each protocol. Since REST services can easily support different content/MIME types we chose this protocol for our communications between clients and the server.

SOAP		REST	
Pros	**Cons**	**Pros**	**Cons**
Support for distributed computations	Heavyweight	Lightweight	No support for distributed computing
Support for HTTP, SMTP, and TCP protocols	Verbose	No need for additional messaging layer	Tied to HTTP protocol
Extensibility	Hard for development	Simpler to develop	Lack of support for security, policy, etc

Table 4: A brief comparison between SOAP and REST protocols.

There are two approaches for the application development in such infrastructures: stateless clients and stateful clients. Stateless clients submit requests to the server for user interactions such as panning the map to a neighboring extent or zooming in/out to a different scale. This method does not maintain state of the data that have been already retrieved on the client, i.e. once they receive a response from the server all objects will be drawn on the map without performing any logic to track these objects. The problem with this approach is that the same objects may be queried again as a result of next requests and considerable amount of network bandwidth will be consumed by each session/instance of the application at the time of these communications. This is not a desired behavior especially in the networks with limited bandwidths such as mobile applications.

Stateful clients have not been the subject of prior tGAP researches hence; we decided to mainly focus on these types of clients and use server-side processes to progressively transfer geometries that are being retrieved from the RDBMS with respect to the generalized data stored in the tGAP index structures.

In the next sections we describe our methodology and propose two different architectures for our stateful client/server applications. Both approaches use local data structures to keep track of the geometries that have been already retrieved on the client and use this knowledge to query the server against the data that have not been requested as a result of previous user interactions. All requests are submitted by a bounding box representing current scale/resolution of the map view on client and a list of edge objects that have been drawn on the map. For the initial phase this list is empty.

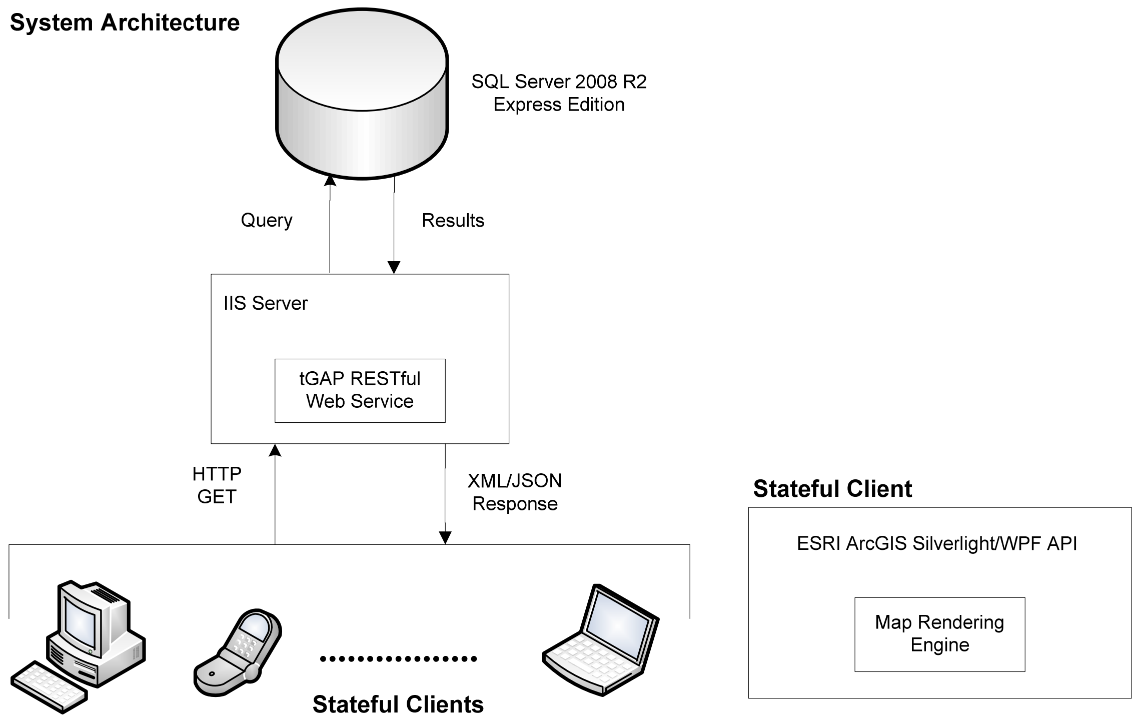


Figure 14: The client/server architecture.

First we choose an efficient method of communication between the two endpoints in order to minimize the amount of data submitted from the server to clients. Then we describe the design for a multithreaded stateful client that polls the geometry information from server that need to be drawn on its current viewport while keeping track of the objects that have been already received. Next, we propose a design for clients that have their server pushing these data to them and perform a methodology comparison between client poll and server push mechanisms. In the end we investigate the effect of database indexes in corresponding spatial queries.

6.5.1. XML vs. JSON

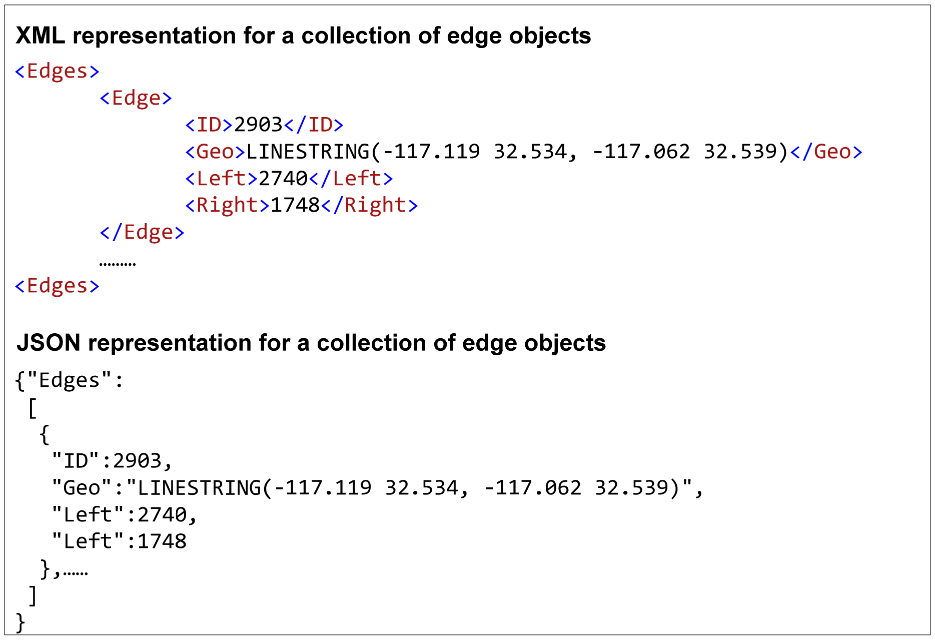
At each request a collection of edge objects will be transferred from server to client. In the context of web services all data submitted between the two endpoints should be serialized first. Data serialization of RESTful services could be done in both XML and JSON formats. In order to minimize usage of the network bandwidth we used the lean tGAP index therefore; tGAP_EdgeID by itself can uniquely identify each edge object. Minimal set of attributes are chosen to serialize data and it contains edge ID with corresponding geometry of the object. We minimized the length of column names to lower the length of serialized data, i.e. ID for tGAP_EdgeID, Geo for Geometry, Left for left face reference ID, and Right for right reference ID attributes and used these attributes to transfer an edge from server to client. Spatial reference information (SRID) is not sent with each geometry since the entire dataset is stored with the same WGS-1984 (4326) projection. This also saves considerable amount of serialized data transferred over the network media.

Geometry instances are stored in compliance with Well-Known Text (WKT) representation of each object as described in Open Geospatial Consortium (OGC). Following examples show how SQL Server 2008 R2 uses OGC definitions of an edge with a line and an edge with a polyline.

- Representation of an edge with two nodes:
  LINESTRING (-117.119 32.534, -117.062 32.539)
  Start node coordinates: (-117.119, 32.534)
  End node coordinates: (-117.062 32.539)
- Representation of an edge with polylines (includes two intermediate nodes in addition to the start and end nodes):
  LINESTRING (-117.062 32.539, -117.039 32.541, -117.027 32.542, -117.026 32.542)

A sample XML and JSON outputs for collection of edges is shown in Figure 15. We compared output of the XML and JSON serializations for the same responses with different number of requested edges. The output sizes are very close with the only difference that for large outputs

JSON performs slightly better (Figure 16). For this reason and the fact that JSON formats are simpler to develop we chose this type of data serialization in our experiments.



XML representation for a collection of edge objects

```
<Edges>
       <Edge>
              <ID>2903</ID>
              <Geo>LINESTRING(-117.119 32.534, -117.062 32.539)</Geo>
              <Left>2740</Left>
              <Right>1748</Right>
       </Edge>
       ………
<Edges>
```

JSON representation for a collection of edge objects

```
{"Edges":
 [
  {
   "ID":2903,
   "Geo":"LINESTRING(-117.119 32.534, -117.062 32.539)",
   "Left":2740,
   "Left":1748
  },……
 ]
}
```

Figure 15: Amount of serialized data submitted from server in XML and JSON formats.

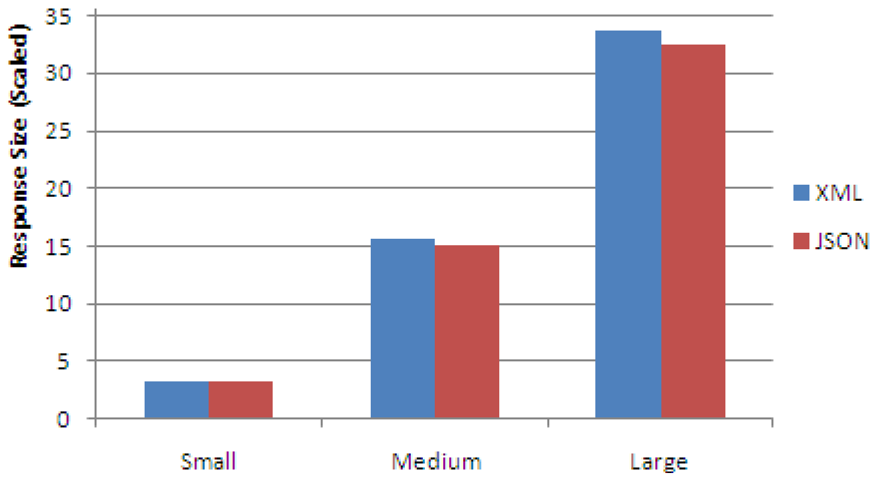


Figure 16: Amount of serialized data submitted from server in XML and JSON formats.

6.5.2. Asynchronous Communications

All requests are submitted asynchronously between clients and the server hence; there are cases where there is a need to cancel an older request which is still in progress on server. Cancellation could be done on client by explicitly invoking a method in the client application or by any action (pan/zoom) that changes current extent of the map (implicitly).

The implementation performs as follows. For each session of the clients that submits a request to the server a worker thread will be instantiated. Then the thread starts fetching edges from the database for the ones that are contained in the input extent. Whenever there is no cancellation request received from the same session of the client, the serialization process of edges retrieved from the database continues.

For implicit cancellation clients listen to two key events of their map views. Whenever the map extent is changing the cancellation request will be submitted to the server right away. This will give server a heads up that a new request is coming and it terminates any ongoing serialization process immediately. Full draw of the map raises the extent changed event and in corresponding handler a new request for the new extent will be submitted to the server.

6.5.3. Maintaining the State

All the clients are stateful meaning that they keep track of the objects received from the server at each interaction of the user. Since each edge can be uniquely identified by its tGAP_EdgeID attribute a list of edge IDs is maintained on the client side. This list is initially empty and will be updated with new edge IDs every time a collection of serialized data received on the client.

For all requests the current map extent on the client application as well as a list of edges that have been retrieved on the client will be passed to the server. Server queries the database against all edge objects in the tGAP index that are contained in the given map extent and removes the edges that are included in the list of edges submitted by the client.

6.5.4. Polling Mechanism (Simplex Communication)

The request/response (polling) is the most common approach used in the client/server environments. In this pattern all requests are initiated by clients over the network media and the server responds to each session with the requested information therefore; clients wait to receive a response from the server.

Each time a user interacts with the map view on the client application a request with respect to the logic described in section 6.5.3 will be submitted to the server. If there is an ongoing request for the same session of the client on the server it will be implicitly cancelled using the method described in section 6.5.2. Server queries the database and builds a collection of <ID, Geo, Left, Right> pairs containing the unique ID, WKT geometry representation of each edge, left and right face reference IDs retrieved from the database. Server serializes these data and responds to the client by a JSON string of requested edges.

After the client receives the response from the server it de-serializes the string received from the server and builds a collection edge objects that correspond to their definition in the Geo attribute. A list of retrieved edges is updated locally on the client every time a response comes from server. Edges will be drawn on the same order they received from the server meaning that the edges with highest importance values will be drawn first.

6.5.5. Pushing Mechanism (Duplex Communication)

In contrast to the polling mechanism duplex communication can be used to push data from server to the client. In this pattern either of the client or the server can initiate the requests and a channel of communication will be associated between the two endpoints through the HTTP protocol or TCP sockets to send and receive the data.

Our duplex communication is implemented as follows. Each time a user interacts with the map view on the client application a request with respect to the logic described in section 6.5.3 will be submitted to the server. If there is an ongoing request for the same session of the client on the server it will be implicitly cancelled using the method described in section 6.5.2. This means that the server stops pushing current (serialized) data to the client and starts fetching edges for the new requested extent and the edges that are not included in the input list. After querying the database server builds a pair of <ID, Geo, Left, Right> containing edge unique ID, WKT geometry representation of each edge, left and right face references of the edge retrieved from the database and creates a queue of grouped edges that belong to each high importance values and serializes the data, i.e. server computes the JSON strings of all edges that belong to the same high importance value (ImportanceHigh attribute) and pushes that small chunk of edges to the client in multiple steps. All serialized grouped edges are submitted with reverse order of their importance values in order to have the client draw more important objects (more generalized edges) first. Each time the client receives a small amount of data from the server and a background thread de-serializes the edges received from the server at the time of each server push. A local collection of edges gets updated every time a set of edges is constructed from their corresponding WKT geometry definitions. Another thread on the client checks for any change in this collection and updates the map view correspondingly hence; users experience smoother map navigation and an overall more responsive application on their end.

Clients developed with this approach work smoother as each time small amount of data (edges with the same high importance value at each sequence of server pushes) will be transferred from the server and drawing operation takes place for small number of instances of the object at each time comparing to a large collection of edge objects in the previous method.

On the other hand since the draw process of maps is expensive it usually takes a little time to finish. During this time a cancellation (explicit/implicit) may be initiated by the request in which requires immediate cancellation of an ongoing draw process in the UI thread.

Nevertheless; these duplex clients are chattier than the clients in the request/response pattern and they require reachable client endpoints in all network communications.

7. Progressive Transfer of Vector Data

In our implementations all the edges of the faces that are partially included in the requested extent are also drawn. This is mainly because there is always high possibility of requiring those edges being retrieved whenever user pans the map to a neighboring extent and/or zooms out the map to a larger scale. The visual results of progressively transferring vector data using our tGAP index structures is presented in Figure 17.

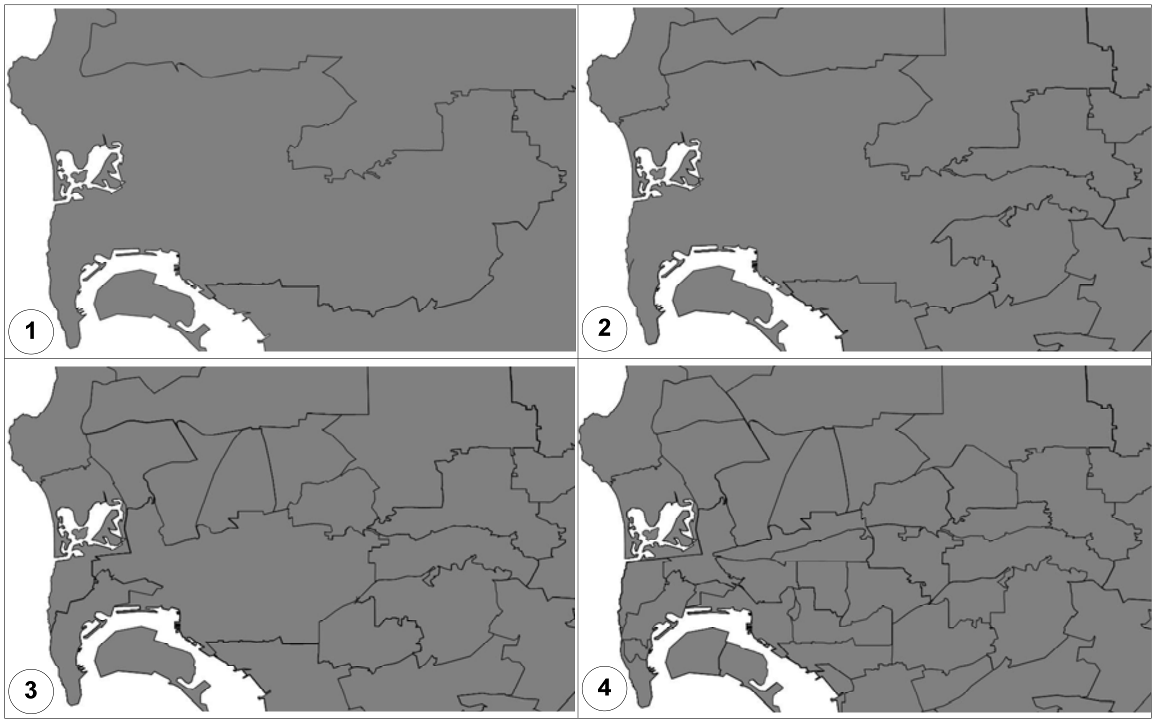


Figure 17: Progressive transfer of San Diego ZIP codes from server to client

Maps show the ZIP codes requested for downtown San Diego, California with the following extent definition:

XMin: -117.284955565711	XMax: -116.920146032504
YMin: 32.6611681268998	YMax: 32.8776527649706

Drawing the edges on the map by themselves builds the visual of corresponding faces. Faces can be easily identified with the left and right references of the data retrieved from the server. We also tried building the faces from these edges. This polygonize procedure causes an extra load on clients as the computations are extensive when there are more edges related to a single face. The steps are as the following:

In the simplest case faces are only related to one edge therefore; one ring will be created based on the edge. For other cases First, a node to edge lookup is created that contains edge start and end nodes as keys. Second, an edge list is used in which all edges that are belonging to the face are stored. An edge is taken from the list, and based on the face that is reconstructed and the side of the edge on which this face lies (left or right), a next edge is taken from the list via the lookup table using the start or end node identifier of the edge. This process continues until the edge list is empty. In case more than one ring can be formed out of the edges, islands are present in a face. After finding all rings, islands rings always have a smaller area than the (largest) outer boundary ring. Thus when looking at the areas of the rings a decision can be made, which of the formed rings must be the outer one. A special case when reconstructing islands is the case of an island touching at the outer boundary ring of a face. Based on the information from the node to edge lookup table, more edge continuations are possible, i.e. the same node in the lookup table points to more edges. This ambiguity can be solved, by using the geometry of the edges and sorting the edges by angle in which they arrive at the node.

7.1. Database Queries

To retrieve the edges that could be drawn on a given map extent we can do the following:

I. Query against the MBR geometry attribute of the faces that overlap the input map extent and join the results with all edges that have these face objects as their left or right face references. Select those edges.

```
SELECT tE.tGAP_EdgeID as ID, E.Geometry as Geo,
tE.LeftFaceID as Left, tE.RightFaceID as Right
FROM tGAP_Face tF
INNER JOIN tGAP_Edge tE ON (tE.LeftFaceID = tF.tGAP_FaceID OR
tE.RightFaceID = tF.tGAP_FaceID)
INNER JOIN Edge E ON E.EdgeID = te.EdgeGeoID
WHERE geometry::STPolyFromText('POLYGON ((-117.329277289258
32.6611681268998, -117.329277289258 32.8776527649706, -116.875824308957
32.8776527649706, -116.875824308957 32.6611681268998, -117.329277289258
32.6611681268998))', 4326).STOverlaps(tF.MBR) = 1
ORDER BY tE.ImportanceHigh DESC;
```

Listing 3: SQL query to retrieve all the edges of San Diego ZIP codes with option I.

Pro: Retrieving complete information for visualization.

Cons: Too many rows with duplicates as a result of the join query between edges and faces, plus an extra join penalty.

II. Query MBR geometry of overlapping face objects with the input map extent then use a separate query to retrieve distinct edges that are contained in aggregated geometry of the above MBR set.

Pro: No double edges retrieved as the second query selects distinct edges.

Con: Like the first approach too many rows are retrieved.

III. Select the edges directly by querying the edges that are contained in the input map extent.

Pro: Performs relatively faster than option II.

Con: Not all the edges of the faces those are partially visible in the current map extent will be retrieved. This issue could be resolved by invoking another query to retrieve all the edges that intersect the input map extent and add them to the above set before the serialization process.

```sql
DECLARE @AggregatedMBR GEOMETRY;
SET @AggregatedMBR = GEOMETRY::STGeomFromText('GEOMETRYCOLLECTION
EMPTY', 4326);
SELECT @AggregatedMBR = @AggregatedMBR.STUnion(tF.MBR)
FROM tGAP_Face tF
WHERE geometry::STPolyFromText('POLYGON ((-117.329277289258
32.6611681268998, -117.329277289258 32.8776527649706, -116.875824308957
32.8776527649706, -116.875824308957 32.6611681268998, -117.329277289258
32.6611681268998))', 4326).STContains(tF.MBR) = 1;

SELECT tE.tGAP_EdgeID as ID, E.Geometry as Geo,
tE.LeftFaceID as Left, tE.RightFaceID as Right
FROM tGAP_Edge tE
INNER JOIN Edge E ON E.EdgeID = te.EdgeGeoID
WHERE @AggregatedMBR.STOverlaps(E.Geometry) = 1
ORDER BY tE.ImportanceHigh DESC;
```

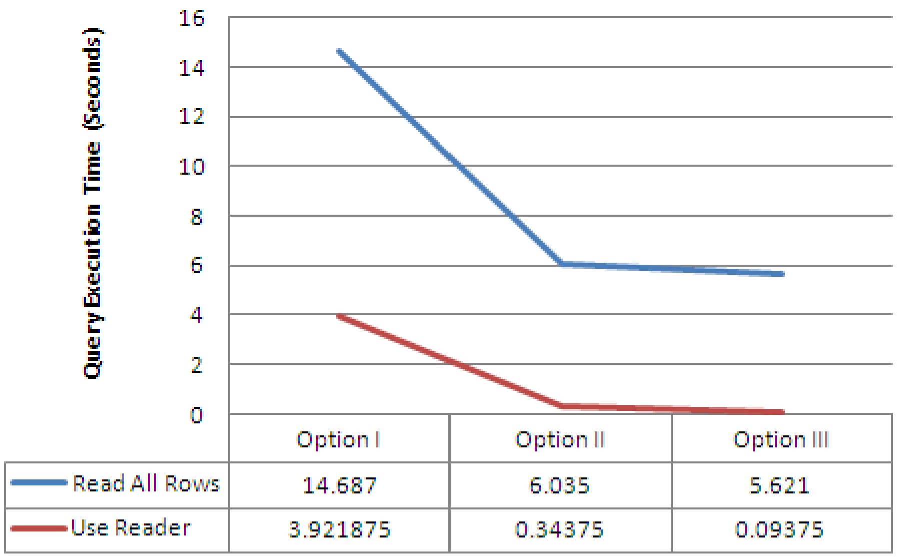
Listing 4: SQL query to retrieve all the edges of San Diego ZIP codes with option II.

```sql
SELECT tE.tGAP_EdgeID as ID, E.Geometry as Geo,
tE.LeftFaceID as Left, tE.RightFaceID as Right
FROM tGAP_Edge tE
INNER JOIN Edge E ON E.EdgeID = te.EdgeGeoID
WHERE geometry::STPolyFromText('POLYGON ((-117.329277289258
32.6611681268998, -117.329277289258 32.8776527649706, -116.875824308957
32.8776527649706, -116.875824308957 32.6611681268998, -117.329277289258
32.6611681268998))', 4326).STContains(E.Geometry) = 1
ORDER BY tE.ImportanceHigh DESC;
```

Listing 5: SQL query to retrieve all the edges of San Diego ZIP codes with option III.

Overall, considering these two execution times will result this option perform slower than the second option.

Figure 18 shows performance evaluation of each option for both methods of reading the entire rows and reading a forward-only stream of rows. The latter allows retrieving the results as soon as they become available in the RDBMS. Both approaches proved that second and third options make a huge difference as using spatial queries to find overlapping faces perform very poor. Either of the second or third options are good candidates to retrieve edge objects on a given map extent. In practice we used the second option as it returns all the required edges.



	Option I	Option II	Option III
Read All Rows	14.687	6.035	5.621
Use Reader	3.921875	0.34375	0.09375

Figure 18: Time needed to retrieve objects with each query option

7.2. Effect of Database Indexes

After analyzing the execution plans reported by the RDBMS we observed that spatial filtering is the most expensive computation step in all the queries.

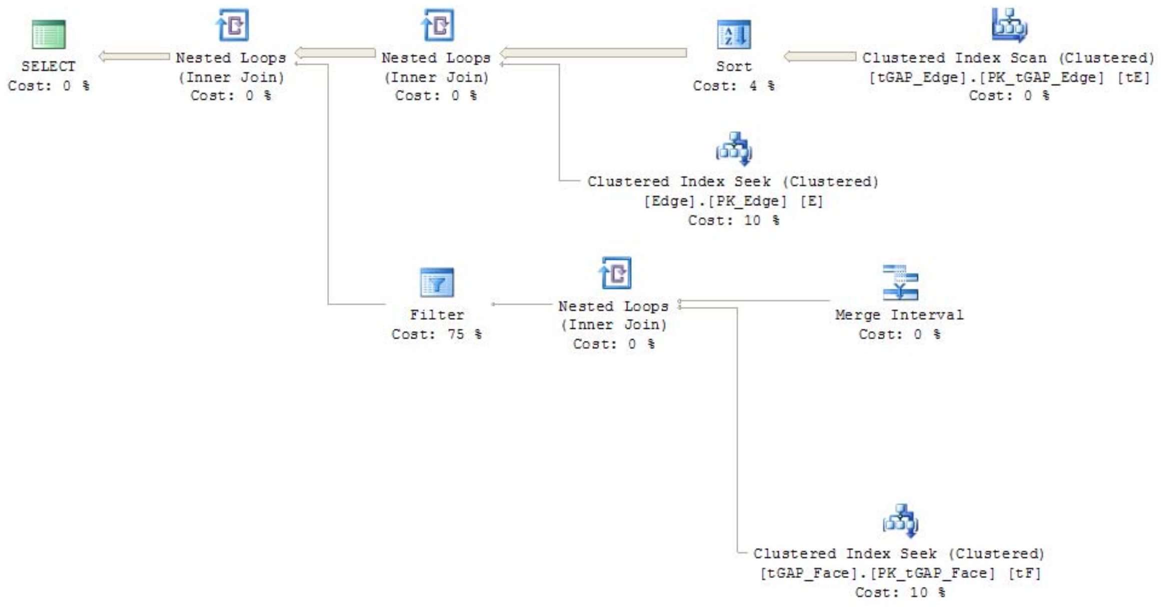


Figure 19: Actual RDBMS execution plan for queries in option I.

Figure 19 shows the spatial filter on tGAP_Face table in option I with 75% of the total query execution time is the most time consuming computation in the query.

The second option is a combination of two separate queries. For both queries the cost of spatial filtering is 80% of the entire process even though the former runs relatively fast. The actual RDBMS execution plan of the second option is depicted in Figure 20.

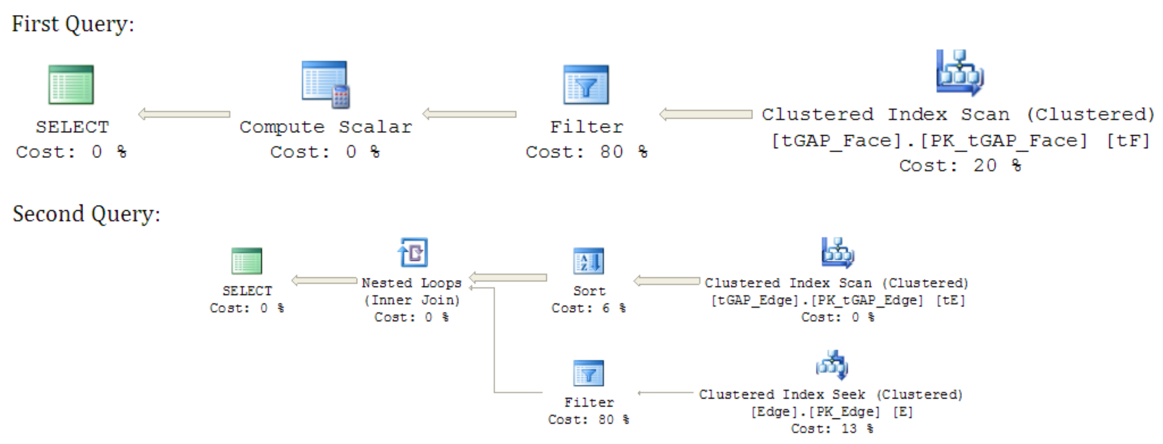


Figure 20: Actual RDBMS execution plan for queries in option II.

Likewise 80% of the processing time for the query in our third option is related to the spatial query of edges (Figure 21).

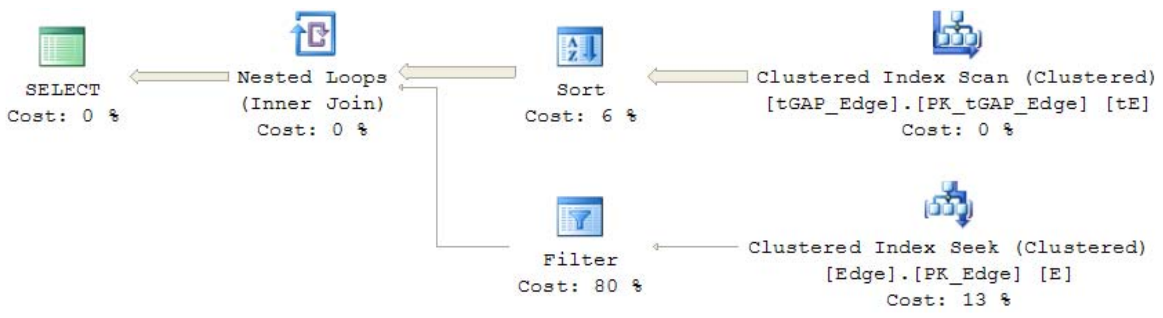


Figure 21: Actual RDBMS execution plan for queries in option III.

All ID attributes are primary keys of our data models, i.e. clustered indexes therefore; any select operation that involves these columns will be fast (10 to 13 percent of the entire process for each table).

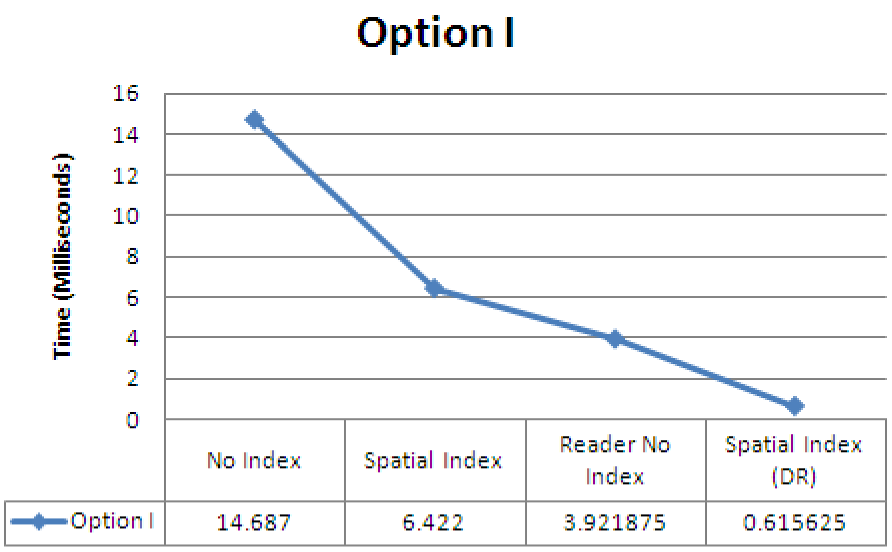
Two spatial attributes are included in our queries: Geometry column from Edge and MBR column from tGAP_Face tables. For our test case we created a spatial reference for the bounding box of the state of California for each of the above columns:

XMin: -124.392637865847	XMax: -114.125230486328
YMin: 32.535781346641	YMax: 42.0021917673185

In addition to attributes that hold geometries some considerations in regards to non-spatial attributes is also necessary. These value based attributes include ImportanceHigh column of the tGAP_Edge table where always a descending order of the query results is required in which retrieves the edges from corresponding table in the database. The LeftFaceID and RightFaceID foreign key attributes in that table are the other two columns that require database indexes.

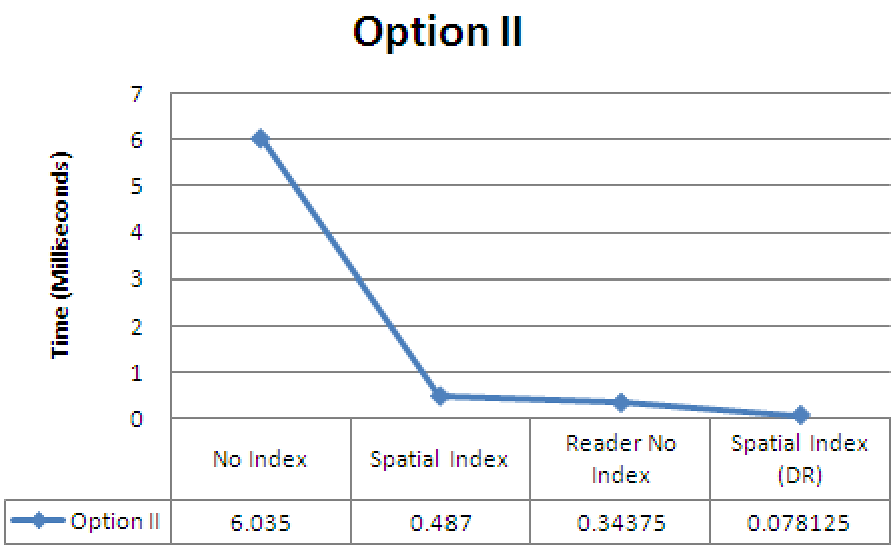
Figure 22 shows the performance optimization of the query with and without attribute and spatial indexes for option I. We gained about 2.3 time of performance increase when retrieving all rows and the same query executed 6.4 times faster when using forward-only stream of rows.

The results of adding database indexes for options II and III are shown in Figures 23 and 24. For second approach the gain in performance is about 12.4 and 4.4 for fetch all and using the data readers respectively. The query of in the third option gained a factor about 17 and 2 for respective data retrieval methods from the database.



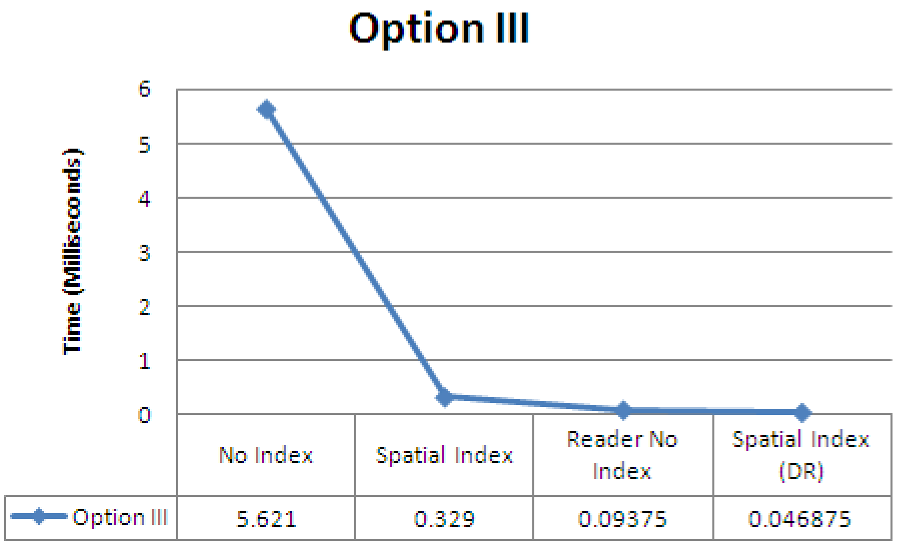
	No Index	Spatial Index	Reader No Index	Spatial Index (DR)
Option I	14.687	6.422	3.921875	0.615625

Figure 22: The Effect of spatial and attribute indexes in option I.



	No Index	Spatial Index	Reader No Index	Spatial Index (DR)
Option II	6.035	0.487	0.34375	0.078125

Figure 23: The Effect of spatial and attribute indexes in option II.



	No Index	Spatial Index	Reader No Index	Spatial Index (DR)
Option III	5.621	0.329	0.09375	0.046875

Figure 24: The Effect of spatial and attribute indexes in option III.

8. Conclusion

This paper presented how the tGAP index structures can be used to progressively transfer vector-based geographic data in client/server architectures. We described variations of this structure and performed methodology comparisons between the original and the lean versions of these indexes. Our experiments show that the lean version of tGAP structures can save the memory storage required to maintain the Edge Forest hierarchical structure by an average compression ratio of 12. Importance values associated with each instance of face and edge objects help retrieval of these objects for incremental updates on each client letting them to show objects progressively on their map views. We proposed two stateful client architectures that both keep track of the objects received at previous requests to minimize the amount of network bandwidth on server responses. In our simplex communication approach a conventional request/response pattern is implemented and clients wait for a collection of serialized objects which are the result of user interactions with the map on the client application. A more complex duplex communication approach is also presented and we showed how a small amount of network bandwidth will be used at each time of the server pushes as only limited number of edges is transferred from the server to the client at lifetime of a request. We showed three different methods to query the edge objects for the requested extent/scale and described advantages and disadvantages of each.

We proposed two recommendations for querying vector data from the RDBMS. Spatial indexed has huge effect in data retrieval of tGAP structures. When fetching the entire result set these indexes improve the performance by an average factor of 15 in our recommended options for retrieval of edge objects.

In summary although tGAP index structures may cause some data redundancies and requirement for more data storage comparing to original datasets they can be effectively used to transfer vector-based geographic objects in the context of client/server architectures especially for the environments with limited network bandwidths.

9. References

[1] The GAP-tree, an approach to “On-the-Fly” Map Generalization of an Area Partitioning, by Peter Van Oosterom. Paper for GISDATA Specialist Meeting on generalization, Compiegne, France (1993)

[2] Variable-scale Topological Data Structures Suitable for Progressive Data Transfer: The GAP-face Tree and GAP-edge Forest, by Peter Van Oosterom (2005)

[3] Constrained tGAP for generalization between scales: The case of Dutch topographic data, by Dilo, A. and van Oosterom, P. and Hofman, A. (2009)

[4] Constrained set-up of the tGAP structure for progressive vector data transfer, Jan-Henrik Haunerta, Arta Dilob, by Peter van Oosterom (2009)

[5] Model Generalization and Methods for Effective Query Processing and Visualization in a Web Service/Client Architecture, by Marian de Vries and Peter van Oosterom (2006)

[6] A storage and transfer efficient data structure for variable scale vector data, by Martijn Meijers, Peter van Oosterom and Wilko Quak (2009)

[7] OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture, http://www.opengeospatial.org/

[8] OpenGIS Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option, http://www.opengeospatial.org/

[9] Detecting and Resolving Size and Proximity Conflicts in the Generalization of Polygonal Maps, by Bader, M. andWeibel, R. (1997)

[10] The tGAP structure: minimizing redundancy and maximizing consistency and offering access at any LoD, by Peter Van Oosterom (2006)

[11] Implementation and testing of variable scale topological data structures Experiences with the GAP-face tree and GAP-edge forest, B.M. Meijers (2006)

[12] World Wide Web Consortium (W3C): http://www.w3.org/

[13] JSON: http://www.json.org/

[14] United States Census Bureau: http://www.census.gov/geo/www/cob/z52000.html

[15] ESRI resource center: http://resources.esri.com/.

[16] The principles of selection, a means of cartographic generalization, by Topfer, F. and Pillewizer,W. (1966)

[17] Database Management Systems – Third Edition, by Ramakrishnan and Gehre. McGraw-Hill (2003)

[18] The Quadtree and Related Hierarchical Data Structures, by Samet H (1984).

[19] CLUSTER RAPTOR: Dynamic Geospatial Imagery Visualisation using Backend Repositories, by Hildebrandt J, Owen M, Hollamby R (2000).

[20] Progressive approximate aggregate queries with a multiresolution tree structure, by Lazaridis I, Mehrotra S (2001)

[21] Remote raster image browsing based on fast content reduction for mobile environments, by Rosenbaum R, Schumann H (2004)

[22] Multi-Representation Databases with Explicitly Modelled Intra-Resolution, Inter-Resolution and Update Relations, by Bobzien, M., Burghardt, D., Petzold, I., Neun, M., and Weibel, R. (2006)

[23] Continuous generalization for fast and smooth visualization on small displays, by Sester, M. and Brenner, C. (2004).

[24] Reactive Data Structures for Geographic Information Systems, by Peter Van Oosterom (1990)

[25] Microsoft SQL Server 2008 R2 Product Documentation: http://technet.microsoft.com/en-us/library/bb418440%28SQL.10%29.aspx