VECTOR ROAD MAP COMPRESSION - A PREDICTION APPROACH

Zongyu Zhang
Georgia Institute of Technology
zongyu@cc.gatech.edu

ABSTRACT

This paper explores a new method to compress vector road network map. The compression schema flows as: (1) Traversing road networks based on its topology; (2) Keeping predicting the next vertex based on the visited vertices; (3) Encoding the prediction errors using the entropy coding method (i.e. Huffman or arithmetical coding). Based on the analysis of the road network’s spanning characteristics, two prediction models will be designed to capture the trends of roads’ “flow” trends. A spanning tree like road network traversal method will be developed to help solve the problem of topological imperfection of digitized road maps and integrate the compression of road network map’s geometrical and topological data. Additionally, a compression benchmark will be designed to help capture the essence of the proposed compression schema and make a fair comparison over the performances of prediction methods. Our prototype implementation has demonstrated that the simple vertex-based linear prediction approach beat the angle length prediction method in reducing the entropy of the predicted vertices’ errors. It also shows the total compression ratio can be close to one eighth for most of the road networks, which agrees with the compression benchmark.

INTRODUCTION

The enormous volume of vector maps, relatively limited communication bandwidths and users’ thirst of instantly response from Web-based Geographical Information System (Web GIS) motivates the need to efficiently transmit vector maps over Internet. In contrast to raster maps, which use image or two dimensional array representations, vector map can be a set of geographical features, such as points, polylines, and polygons. A polyline is a list of consecutively connected points. A polygon is a closed list of consecutively connected points. Compression techniques for vector maps allow users to retrieve or store extra information, such as the owner of the house, address book, road name etc within the same resources’ limitation. With the fast evolving computational capabilities of people’s terminal devices (workstation, PDA, etc), it would be more efficient to compress map before transmission.

Generally, the compression techniques can be divided into two groups, one is the lossy compression. The other is the lossless compression. The former simplifies the map features before compression. The key issue is how much error the simplification will generate. Lossy compression schemes can be acceptable in applications, which do not have tight requirements over the locations of spatial objects. For example, people may only need to figure out the orientations of roads during urban navigation. Most of researchers try to achieve best approximations in their lossy compression [1] in terms of the maps’ specific application domain.

This paper explores the problem of using entropy-encoding approach to achieve lossless vector road map compression. It proposes the use of the vertex prediction method to make a guess of the next vertex based on the road networks’ characteristics. Then an error distribution based method will be used to compress the prediction errors. The experiments exposed by choosing appropriate prediction and road network traversal methods, the bits needed to encode each vertex can be much smaller than the original representation.

Prior Art

Map compression techniques can be divided into two groups, namely raster map compression and vector map compression. Raster map compression techniques manipulate a raster matrix to get a concise representation.

Vector maps consist of points (vertex also), polylines and polygons. Typically, road network maps are composed of inter-connected polylines. The vertex at both ends of the polyline is called end-node. Those inner polyline vertices are called inner-node. A road segment is the part of the road which lies between two crossing roads. Generally, each road segment should be corresponding to one polyline. However, in real dataset, it can be stored in several
consecutively connected polylines.

Polylines can be unredundantly represented by three tables in GIS. One table lists the Polyline’s ID\((pID)\)--\(PTable\), the starting node’s ID \((sID)\)--\(ITable\) and the ending node’s ID \((eID)\)--\(ETable\). The second table lists the tuples of \(pID\) and its ordered list of inner-nodes’ coordinates \((x,y)\). The third table lists the end-nodes’ IDs and the corresponding coordinates.

![Figure 1. Example polylines.](image)

<table>
<thead>
<tr>
<th>(pID)</th>
<th>(sID)</th>
<th>(eID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>1n</td>
</tr>
<tr>
<td>2</td>
<td>1n</td>
<td>2m</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>1n</td>
</tr>
</tbody>
</table>

**Table 1. PTable**

<table>
<thead>
<tr>
<th>(pID)</th>
<th>Inner-nodes’ coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((x_{11}, y_{11}), (x_{12}, y_{12}), \ldots, (x_{1n-1}, y_{1n-1}))</td>
</tr>
<tr>
<td>2</td>
<td>((x_{21}, y_{21}), (x_{22}, y_{22}), \ldots, (x_{2m-1}, y_{2m-1}))</td>
</tr>
<tr>
<td>3</td>
<td>((x_{31}, y_{31}), (x_{32}, y_{32}), \ldots, (x_{3l-1}, y_{3l-1}))</td>
</tr>
</tbody>
</table>

**Table 2. ITable**

<table>
<thead>
<tr>
<th>(Node\ id)</th>
<th>Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>((X_{10}, Y_{10}))</td>
</tr>
<tr>
<td>1n</td>
<td>((X_{1n}, Y_{1n}))</td>
</tr>
<tr>
<td>2m</td>
<td>((X_{2m}, Y_{2m}))</td>
</tr>
<tr>
<td>30</td>
<td>((X_{30}, Y_{30}))</td>
</tr>
</tbody>
</table>
The ID numbers of above tables are integers (4bytes). The coordinate of each node is a tuple of double precision (8bytes). The topology information of the polyline is represented by the IDs of the three tables. The geometry part is stored in the $I_{\text{Table}}$ and $E_{\text{Table}}$ as coordinates. The size of the whole vector road network map is the sum of all the above tables’ size. Given the number of polylines ($#\text{PS}$), average length of polylines (aveLen), the total number of end nodes ($#\text{EndNode}$), and the decimal precision of each node ($#\text{unitBytes}$). The $#\text{unitBytes}$ is normally 8 (float tuples) or 16 (double tuples).

- $P_{\text{Table}}$: $#\text{PS} \times 12$
- $I_{\text{Table}}$: $#\text{PS} \times [4 + (\text{aveLen}-2) \times #\text{unitBytes}]$
- $E_{\text{Table}}$: $#\text{PS} \times #\text{EndNode} \times (#\text{unitBytes}+4)$

Normally, there are much more inner-nodes than end-nodes in road segments. That makes aveLen much greater than $#\text{EndNode}$ and also means the size of the inner-nodes plays much more important than the size of end-nodes and topological information.

Vector map compression includes techniques such as feature simplification [2], chain codes, dictionary based compression [1], and feature approximation [3]. Feature simplification [4,5,6] and chain codes [7] are used with paper maps and other line drawings containing curves. These techniques approximate curves by a sequence of straight-line. They use straight lines from a fixed collection, e.g. vertical, horizontal, diagonal. Feature simplification and chain-codes often eliminate original points and add new points as a side effect. Dictionary based algorithm construct a dictionary to match dictionary entries to line segments. When a dataset is decompressed, the dictionary is searched to find the data associated with the distribution (slopes and offsets of segments in the curves). If the distribution of line segments in a dataset does not match well with the dictionary entries, the errors of approximation can be large.

Most of the recent vector map compression techniques treat the topology and geometry separately [1]. The compression schema only works for geometrical data. After transmission, the geometry data will be combined with topological data to produce the correct map. This approach is reasonable due to the fact that in vector map, there are much more inner-nodes than the end nodes. That makes the topological data much smaller than the geometrical part. Researchers [1,6] try to simplify the vector map before compression or transmission. In [1], all $(\Delta x, \Delta y)$ s (i.e. the tuples of $x_i-x_0$ and $y_j-y_0$) is starting node, and $(x_i,y_i)$ is inner-node, $0 \leq i \leq n-1$) were clustered and got 256 typical representatives. The starting node was viewed as the based point. Each inner-node was substituted by its corresponding $(\Delta x, \Delta y)$ to its based node. Thus, each inner-node of polylines will be encoded as a one-byte integer (0–255). That saves much more space compared with encode inner-nodes in double/float types. However, if the average length of road segments is not big enough, that topology information will occupy almost the same space with the geometrical part after compression.

In the area of the 3D geometry compression, people pay more attention to the compression of topological data [10,11], which is also called connectivity. All vertices are traversed in a predefined traversal order and encoded as a stream of symbols that describes the operations used to grow the growing region over the triangle mesh. Efficient coding of the symbol stream such as Huffman coding or arithmetic coding allows coding the connectivity with often less than 2 bits per vertex and never more than 4 bits per vertex.

Most of the prediction methods only follow some simple geometry relationships, such as linear and parallel [12]. Inspired by their work, this paper will explore both topological compression and prediction methods for the purpose of road network vector map compression.

![Figure 2. Map Compression Framework.](image-url)
Road Network Map Compression Framework

The framework of the proposed compression schema tries to achieve the compression of both topological and the geometrical information.

VECTOR ROAD NETWORK MAP AND ITS COMPRESSION

As discussed before, road network maps consists of connected polylines. The connections between road segments forms the topological properties such as crossing, closed, adjacent, and disjoint objects. A map also has a scale and usually supports operations such as zoom in or zoom out.

Facts of Vector Road Network Map

With the fast development of Geography Information System (GIS) and Internet technologies, institutions, i.e. government, or research centers, tends to share their GIS datasets. Most of the states have their own GIS data clearinghouse, which provides Internet access to those data. In this paper, we apply the data downloaded from the National Atlas of the United States of America, which provides geographic data in shapefile format. It contains the major road and highway from the United States Atlas[4]. Shapefile is a data format developed by ESRI company. In this file, each vertex is a tuple of double values (16bytes). Normally, those road data was initially digitized from the aero photos. Some of them can also be obtained by field survey.

As one kind of man-made feature, roads try to keep as smooth or straight as possible. That observation is the basis to develop the prediction method. Furthermore, by analyzing the road maps, we can find particular spatial characteristics of different kinds of the roads. That would provide us the hint to develop appropriate prediction model for specific kind of roads.

From the comparison table, we can see that the highways mostly have good smoothness. If we connect those tiny segments of highways, we can get some group of extended long highways, which can have smoothness and straightness similar to the interstate highway segments. With each segments of highways, there mostly exists two kinds of pattern between the road segments: they can be either in a “loop” style or in a “zigzag” one. Due to the practice of cartography, people would break a straight line into smaller segments except that it intersects with other roads. Semantically, there should be polyline only exists between road intersections.

In the real dataset from GIS database, however, roads may be broken into some arbitrary order of tiny polylines even without intersection with other roads, as demonstrated in figure 3. In this figure, each arrow represents one piece of polyline. The whole road segment was represented by five connected polylines. For the conciseness of demonstration, the inner-node of each polyline was omitted in figure 1. This phenomenon is very obvious in the streets of a city area. Those observations motivate us to develop the following prediction methods.

A 2D polyline is composed an ordered list of 2D vertices. In the following we use the following notation: \( V = \{v_0, ..., v_n\} \) is the set of vertices \( v_i \), where \( v_i \) is in \( R^2 \). Then, a polyline \( P \) is defined by \( P = \{v_i|0 \leq i \leq n-1, v_i \) is sequentially ordered from \( v_0 \) to \( v_{n-1}\} \) after connecting and ordering its component sub-strings.

Vertex Based Linear Prediction Model

In this model, we try to predict the next vertex \( v_3 \) based on its previous two vertices \( v_1 \) and \( v_2 \). The prediction model can be simply formulated as: \( v_{err} = v_3 - (2*v_2 - v_1) \). \( v_{err} \) is the error produced by the prediction model. This prediction model assumes Vector \( v_3v_2 \) and \( v_2v_1 \) are equal. Using this prediction model, each polyline should record at least two points as the initialization.

Angle-Length Based Prediction Model

This prediction model can be demonstrated by the following graph. It will use four vertices as the basis. It assumes the turn angle of road segments can be expressed in the following linear relationship:
The above angle based prediction model predicts both the turn angle for next vertex and the length of the next turn. The prediction errors will be stored.

**Entropy Encoding**

An *entropy encoding* is a coding scheme that involves assigning codes to symbols so as to match code lengths with the probabilities of the symbols. Typically, entropy encoders are used to compress data by replacing symbols represented by equal-length codes with symbols represented by codes proportional to the negative logarithm of the probability. Therefore, the most common symbols use the shortest codes.

According to Shannon's theorem \[13\], the optimal code length for a symbol is \(-\log_b P\), where \(b\) is the number of symbols used to make output codes and \(P\) is the probability of the input symbol. Normally, we set \(b\) to be 2 if using binary encoding. The average code length for a symbol was defined as *entropy*:

\[
e = -\sum_{i=1}^{n} (P_i \cdot \log_b P_i)
\]

Two of the most common entropy encoding techniques are Huffman encoding and arithmetic encoding.

A straightforward encoding of the polyline as a list of vertices includes a lot of redundant information. So, our goal was to transform the data into a representation with less redundancy. Or in other words, we try to reduce the entropy of a symbol that has to be encoded. In the original polyline representation, the symbols that have to be encoded are \((v_i, v_{i+1})\). So, we have \(n+1\) different symbols. The entropy of a symbol would be \(\log(n+1)\). We tried to use geometric relations between neighbor vertices to reduce the entropy of an encoded symbol. Possible relations between vertices include distance and direction, for example, our lossless compression approach is based on four main steps: normalization, quantization, prediction and error encoding.

**Vector Road Network Map Compression Benchmark**

A benchmark was designed to evaluate the performance of the compression. The benchmark should also be independent of the particular data format to store the original data. As mentioned before, the compression tries to achieve small entropy for errors. However, not all vertices can be predicted. Each predictor has to have a minimum sequence of vertices as the predictor initialization. For example, the simple vertex based linear prediction requires the first two vertices as the base and the angle based linear prediction requires the first four vertices as the base. Those lines, which have fewer vertexes than the base, cannot be compressed. They should be encoded directly as tuples of float values. The base nodes required by the prediction model should also be encoded directly as tuples of float values.

The errors generated by the prediction process will be encoded based on their distribution using either Huffman or arithmetic encoding.

The looking up dictionary for prediction errors is composed by the distinct error values and their corresponding probabilities.
Using Huffman coding method, this part can be expressed in a binary tree structure. Suppose, the average length of the polylines is \( l \). The entropy of the error vectors is \( e \) bits. The total number of Vertices is \( V \), the total number of polylines is \( L \) and the total number of end-nodes is \( t \). In the simplest representation, the line strings will be composed of sequential indexed vertices. The coordinates of the line should be in float point precision and the index number can be in integers. Thus, the size of the original data can be roughly estimated as:

\[
S = L(\ell \times 8 + t \times 12) \text{ bytes}
\]

The size of the compressed file can be estimated as:

\[
Z = c + (b + a) \times 8 + \frac{e \times d + t \times sn}{8} \text{ bytes}
\]

The compression ratio is \( Z/S \). As discussed before, if our prediction model works well, the size of \( c \) can be very relatively small compared with the compressed vertices. In terms of a given prediction model, \( a \) and \( b \) are fixed values. They expose how much data cannot be compressed by our prediction model. Presumably, this portion of data can be very small, because the road networks has much more inner nodes than the end nodes. If the average length of the polylines is much bigger than the base length, the value of \( d \) will be very close to value of \( L \cdot l \) and the compression ratio will mostly depends on value of \( e \). Otherwise, we are going to pay too much into the short lines. The value of \( e \) determines the performance of our prediction model.

The above analysis exposes the fact that the prediction-based compression prefers few long polylines instead of lots of short ones.

**Implementation**

In the first step we normalize the coordinates of vertices, so that \( v_i \in [0,1] \times [0,1] \). In the second step we quantize these values. After this step, all our vertices have integer coordinates. Thus, we have \( v_i \in [0,2^B-1] \), where \( B \) is the quantization factor. By choosing a small \( B \), we are losing information. The next step is the core of the whole algorithm. That is road network traversing and vertices prediction. In this step, we try to use the geometric relation between neighbor vertices to produce symbols with less entropy than the original \( v_i \)’s. Based on \( v_i, v_{i-1}, \ldots, v_{i-j} \) we predict the coordinates of \( v_{i+1} \). Then, we compute an error vector \( \Delta_i \) between the true vertex and predicted one. These \( \Delta_i \)’s built the new symbols that had to be encoded. If the predictor could well capture the next vertex, a lot of these difference vectors are expected to be similar, which reduces the entropy of a symbol. Finally, we used a Huffman coder to encode our symbols \( \Delta_i \). The encoded difference and a header are written to a binary file. The header contains information about the bounding box, the quantization factor and the Huffman tree that is used as a dictionary to decode the symbols. We also have to include the several vertices in the header part of each polyline.

Instead of separating the topological data from the geometrical one during the compression, a spanning tree based traversal method is utilized here to help the reconstruction of the topological relationship. As mentioned in the analysis of the road network map, there can be lots of semantically imperfect polylines in our dataset. The following traversal method helps to maintain the semantics of polylines.

For the convenience of implementation, we assign each polyline a direction. It should be consistent with the order of the vertices list. That makes each polyline has a from-node and to-node. The end-nodes (from-node and to-node) of polyline have the list of pointers to its connected polylines. During the road network traversing, a queue is maintained to buffer all the to be visited lines. Every polyline has a pointer field, which stores the address of its “parent polyline”. First, the traverser will pick a startline, which has vertices more than the requirements of predictor from the polyline pool, compress it and push it into the working-queue. The traverser keeps removing the front node from the polyline working-queue and use it as the working polyline. Then, get the connected polylines by the working polyline’s from-node and to-node. Name them as encountered lines. A specific queue (end-queue) was defined to store the base nodes starting from the from-node of the starting polyline. We defined four symbols to indicate the topological relationship between the working polyline and the current encountered polyline.

“S”—the encountered line has never been traversed and now it is processing

“L”—the encountered line is connected with the from-node of working line

“R”—the encountered line is connected with the to-node of working line

“E”—the working-queue is empty and all the encountered lines has been traversed. The “E” symbol indicates the
ending of the spanning tree.

The traversing algorithm can be briefly described as following:

Required data structures:

- CtrlVerNum – the minimum number of vertices required by the predictor to predict the next one
- Queues: working-queue, end-queue as defined before
- Working-line: the polyline dequeued from the working-queue.
- Encountered-line: the polyline connected with the working-line.
- Pred-queue: the queue to aid predictor

Two aiding procedures:

- FillingPredQueue(polyline curline): flushing the pred_queue and filling the emptied pred-queue starting from the from-node of curline. It keeps padding the pred_queue by looping through the parent pointer till the pred_queue has CtrlVerNum vertices.
- ReversePolyline(polyline curline): reverse the vertices order of curline.

Traversal process:

While(visited lines number < total line number)

1. Get one polyline from road network, which length is greater than CtrlVerNum. Write down a “S” and write down the first CtrlVerNum vertices. If its length is greater than CtrlVerNum, applying the predictor and keeping writing down the prediction errors till the end of this line, enter this line into the working queue;
2. Empty the end-queue, put the first CtrlVerNum vertices into end-queue by reversely order;
3. while(working-queue is not empty)
   a. Dequeue a working-line from the working-queue
   b. Get the connected polylines from working-line’s from-node.
      i. Given a unvisited line cline
      ii. If cline does not connect with working-line by cline’s from-node, reverse cline.
      iii. FillingPredQueue(working-line).
      iv. Setting cline->parent to be working-line.
      v. Writing down an “L”, and using the pred_queue to make prediction for all the vertices of this line, writing down the prediction errors.
      vi. Entering this line into the working-queue.
   c. Get the connected polylines from working-line’s to-node.
      i. Given a unvisited line cline
      ii. If cline does not connect with working-line by cline’s from-node, reverse cline.
      iii. FillingPredQueue(working-line).
      iv. Setting cline->parent to be working-line
      v. Writing down a “R”, and Using the Pred-queue to make prediction for all the vertices of this line and write down the prediction errors.
      vi. Entering this line into the working-queue.
   d. Write down the “E” to end this spanning tree.
4. End While;
End While.

Table 4 gives a simplified road network. For the concise of representation, all the inter-nodes of each polylines are omitted. The arrow symbolized the direction of polyline 1. Suppose we start from polyline #1. The recorded symbols are:

S1L3L7R2R5ER4ER8R9EEER6EEEEE
For the concise of representation, all the delta nodes are omitted in the above coding string.

The above table also shows how the polyline orders will be changed during the traversing. All the polylines will be ordered to “flow out” of the starting line. Correspondingly, there would be much more “R” in the coding string.

The above traversal process brings the additional overhead of new symbols into the compressed file. There are only four symbols for the traversal purpose. If applying the angle based prediction model, we should also need to add another two symbols to indicate the clockwise of the angle error. That totally brings six additional symbols. Empirically, the “R”, “E”, and clockwise symbols dominate. That is to say, we only need around two additional bits for each predicted vertex. This portion is much smaller than the error codes themselves and will not be counted in calculating the compression benchmarks.

The traversal process visited each polyline exactly once. In the recovery process, the decoder will add vertices and polylines into the road network according to the meanings of corresponding symbols. If reading “S”, the decoder will create a new road network and interpolating the starting line from the proceeding data. If reading “L”, the decoder will add the proceeding compressed line to the “front” of the starting line. If reading “R”, the decoder will add it to the “end” of the starting line. “E” ends the role of the current starting line.

**EXPERIMENTS AND RESULT ANALYSIS**

**The Experimental Data**

The dataset included 12,911 road segments from the National Atlas of the United States of America [14], which provides geographic data in shapefile format. The shapefiles were converted to Arc/Info [9] coverages using the ArcToolbox. Spatial attributes (coordinates) of the coverages were then extracted in a text file using the Arc command “ungenerate”. Totally, we got 110,771 distinct vertices. The average length of the experimented dataset is around 10.

![Experimental data](image)

**Figure 5.** Experimental data.

![Polyline length distribution](image)

**Figure 6.** The polyline length distribution.

In this data set, there are around 20% polylines, which length is 2 and 10% polylines, which length is 3. Based on
the benchmark, if applying the linear vertex based predictor for our compression, 20% portion vertices will be written into file without compression. The portion of vertices needs to write into file without compression by using the angle based predictor will be around 41%. That portion of the cost makes the compression almost of nonsense.

However, as discussed before, by connecting the tiny segments into much longer polylines, we can reduce the overhead of write down too many uncompressed short polylines. In this case, the cost of the compression will mostly depends on the size of dictionary and the entropy of prediction error. Those two factors will only depend on the performance of predictors. Following the previous description of the road network traversal mechanism, we should optimize the traversal sequence to minimize the prediction errors and achieve less entropy. The prediction errors for those vertices within long polylines will always be counted into the prediction. For those major roads, the tiny polyline segments is mostly caused by intersected with other roads. The inner nodes within those polyline segments may be predicted in multiple ways because the traverser has multiple choices on branching at the intersection node. Thus, the predictor cannot have the global knowledge of the prediction error distribution before the end of compression. Here, the author applies a heuristic method to approximate such a distribution by allowing the traverser using an adaptive procedure to choose the best branch at intersection node.

Presumably, the distribution has a normal form: 

\[ p(err = er_i) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(er_i - \mu)^2}{2\sigma^2}} \]

At the beginning, \( \mu \) is set to be 0 and \( \sigma \) to be 1.

The predictor keeps updating the \( \mu \) and \( \sigma \) by the predicted errors' mean and variances after each prediction. At the intersection point, the next point within all possible branches will be predicted. The traverser will choose the branch proportionally to the value of \( p(err = er_i) \), i.e. those predictions, which have higher error probabilities, will have more chances to be chosen as the next compressing road segments. The above branch-choosing schema prefers small prediction errors. The branch-choosing schema also has the effect to condense the spreading of prediction errors. That helps reduce the entropy of prediction errors.

Compression Results and Analysis

The original experimental road network data was stored in shapefile, which stores each point in a tuple of double values. The shapefile specification is targeted to fit the requirements of high precision mapping. Our experimental road network does not have such fine-grained resolution. The Major Roads of the United States map layer shows the major roads and ferry crossings in the United States, Puerto Rico, and the U.S. Virgin Islands that can be represented at a map scale of 1,200,000 (1 inch on a map at that scale equals about 31.6 miles on the land surface). In binary, the map scale should be within 21 bits. Practically, the experimental road network can be well captured by float value (32bits) rather than double. The original shapefile stores geometry data in two types of files: *.shx stores the indices for vertices' ID and the road segments' connection; *.shp stores the vertices' coordinates in double precision. The size of the vector road network should be estimated by summing the *.shx file size and half of the *.shp file size. The value is 1258kbytes.

The vertex based linear prediction approach generates 380 distinctive \( Xerr \) and 369 distinctive \( Yerr \). The entropy of \( Xerr \) is 4.91 and the one of \( Yerr \) is 4.92. The total number of topological signs used by traverser is 25820.

The above signs were also encoded based on the Huff-man coding schema. The decoder should be able to distinguish the sign code from the \( Xerr \) code. Compared to the number of prediction errors, the one of the signs is much smaller. Simply, we can just plug the shortest code of the \( Xerr \) at the front of the sign codes. The most frequent \( Xerr \) is of frequency 0.108769. Its code length is 4. If the code is “1010”, then we can have the following coding table for signs.
Table 5. The Sign Encoding Results

<table>
<thead>
<tr>
<th>Sign</th>
<th>Percent (%)</th>
<th>Coding Length (bits)</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>“S”</td>
<td>5</td>
<td>7</td>
<td>“1010111”</td>
</tr>
<tr>
<td>“L”</td>
<td>7</td>
<td>7</td>
<td>“1010110”</td>
</tr>
<tr>
<td>“R”</td>
<td>42</td>
<td>6</td>
<td>“101010”</td>
</tr>
<tr>
<td>“E”</td>
<td>46</td>
<td>5</td>
<td>“10100”</td>
</tr>
</tbody>
</table>

The compressed file can be divided into two parts. The first part is the header file. It includes the map range (four floating points), factoring factor (an integer), sign-code looking up table, and the prediction error-code dictionary. The second part is the string of coded symbols.

The above table exposed that the vertex based linear prediction outperforms angle-length based prediction with less dictionary size and much less entropy.

Table 7 concludes the performance of the compression schema over both road network’s both topological and geometrical data. If applying the compressed file size is 161 kbytes. The compression ratio is 12.8%. Shekhar. et al [1] could compress the road network into 365 kbytes. The ratio of table 8 represents how much the topological/geometrical info occupies correspondingly in the compressed file.

Shekhar. et al [1] presented the latest available result on vector road map compression. They could only achieve 29% compression ratio without reducing the size of the topological part. They use one byte to represent each inner-nodes by data clustering. However, they spent much more space in storing the coordinates of the end-nodes and topological info. As table 7 demonstrated, the prediction-based approach spend 10 bits to represent each inner-node. The traverser and the recovery schema effectively reduce the size of the topological part. The entropy encoding is a lossless compression schema. Thus, the prediction-based compression won’t generate errors during compression procedure. While on the result graph of [1], reader can visually identify those errors.

The traverser encoded the topological information into signs. By interpolating the sign string, the decoder is able to recover the road map. However, it is not able to recover the Ptable. That is because the road branches may converge at somewhere. The converged node will be predicted and compressed from all reachable branches. To recover Ptable, the decoder should maintain a lookup table for the end-nodes of polylines. That table only keeps one record for all points, which have the same coordinates. The lookup table can be implemented by using a binary tree.
CONCLUSION AND FUTURE WORK

The experiments demonstrated that the simple vertex based linear prediction performs better in compressing vector road maps. In terms of encoding for the predicted vertices, the vertex based linear prediction method can achieve almost 8:1 compression ratio.

By applying the traversal method, we can link tiny road segments into longer one. This process should reduce the polylines’ number to be much smaller than that of the original files. It reduces the ratio of uncompressed vertices to be much smaller. As discussed in the compression benchmark, the encoding entropy will determine the overall compression ratio. Optimally, that value can be very close to 8:1.

In this article, we only applied the simple linear prediction model. By combining the method used here with other prediction models, even higher compression ratios are possible.

The proposed method can also be applied into the lossy compression by integrating with a polyline simplification method. Try to keep the simplification be consistent with the prediction models, we might achieve better compression ratio.

The compression schema visits each polyline once during the network traversing. That bounds the computation complexity to be within the number of line segments. The predictors work by linear operations. Entropy encoding, such as Huffman coding, are standard process and can be easily implemented. The decoding schema has the same computation complexity as the encoding one. The computation complexity of the proposed vector road map compression method only scales with the number of the road segments.

REFERENCES