COMA: Road Network Compression For Map-Matching

Abdeltawab M. Hendawi[1,2] Amruta Khot[2] Aqeel Rustum[2]
Anas Basalamah[3] Ankur Teredesai[2] Mohamed Ali[2]

[1]*Department of Computer Science and Engineering, University of Minnesota, MN, USA*
[1]{hendawi}@cs.umn.edu
[2]*Center for Data Science, Institute of Technology, University of Washington, WA, USA*
[2]{ hendawi, akhot, binrusas, ankurt, mhali}@uw.edu
[3]*Computer Engineering Department and KACST GIS Technology Innovation Center*
Umm Al-Qura University, Makkah, Saudi Arabia
[3]ambasalamah@uqu.edu.sa

***Abstract*—Road-network data compression reduces the size of the network to occupy lesser storage with the aim to fit small form-factor routing devices, mobile devices, or embedded systems. Compression (1) reduces the storage cost of memory and disks, and (2) reduces the I/O and communication overhead. There are several road network compression techniques proposed in literature. These techniques are evaluated by their compression ratios. However, none of these techniques takes into consideration the possibility that the generated compressed data can be used directly in map-matching. Map-matching is an essential component of routing services that matches a measured latitude and longitude of an object to an edge in the road network graph. In this paper, we propose a novel compression technique, named *COMA*, that significantly reduces the size of a given road network data. Another advantage of the proposed technique is that it enables the generated compressed road network graph to be used directly in map-matching without a need to decompress it beforehand. *COMA* smartly deletes those nodes and edges that will not affect neither the graph connectivity nor the accuracy of map-matching objects' location. *COMA* is equipped with an adjustable parameter, termed *conflict factor* $\mathcal{C}$, by which location-based services can achieve a trade-off between the compression gain and map-matching accuracy. Extensive experimental evaluation on real road network data demonstrates competitive performance on compression-ratio and the high map-matching accuracy achieved by the proposed technique.**

I. Introduction

Extensive availability of GPS-enabled devices has increased the need for routing and navigation services. The storage and transmission of road-network data is the biggest performance issue facing such services and is an important data management challenge. Road-network map, road map for short, is represented as a graph structure with a set of nodes, edges and edges weights, i.e., travel distance or time. To provide a navigation service, the user's location, as measured by a GPS device, is *continuously* map-matched to an edge in the graph. This edge represents the current road segment that the user is believed to be travelling on.

Map-matching links an object location, i.e., latitude and longitude coordinates, to the corresponding edge in the underlying road map [13]. Map-matching is crucial for location aware services that answer queries based on the current and/or future objects' location [5], [6]. Traditionally, map-matching is performed on the original (i.e., non-compressed) road network data. For example, an in-car GPS device stores the digital map of the commuted area, i.e., city, state or country, such that the car location can be mapped correctly to a road segment in this map. However, there are several situations and application scenarios where a compressed version of the road network data is appreciated.

Map compression enables small size devices, e.g., smart watches and navigation drones, to carry the road map for large areas. More specifically, compact representations of road map data are triggered by the need to: (a) reduce the cost of storage devices, e.g., Solid State Drive (SSD), (b) reduce the I/O overheads, and (c) cut down the communication cost and battery consumption in the case that the road map is stored on the server side and is transmitted to the client side over the network.

Motivated by the above reasons, road-network compression becomes an essential goal to spatial database researchers. In fact, there are several compression techniques proposed in literature [1], [7], [9], [15], [17]. These techniques strive for a high compression ratio as its major performance measure. However, none of these techniques focus on the quality of map-matching on the generated compressed data. Moreover, the compressed map generated by some of these techniques cannot be used directly to perform map-matching without an initial phase of decompression to restore the original form of the map. This initial phase leads to high CPU power wasted in decompression of the compressed map and, hence, increases battery consumption. Furthermore, in some lossy compression techniques, the compressed version of the road-map is not an equivalent representation of the original one. Some of the map details are lost during the compression process. The quality of lossy compression techniques are evaluated based on visual similarity or dissimilarity between the generated map (after compression) and the original version of the map (that is before compression). While visual similarity is a valid measure of performance in some applications, we set our performance measure to be the quality of map-matching using the compressed version of the map. Losing some critical information such as the exact locations of specific nodes (e.g., intersections and highway exits) leads to low accuracy in the map-matching results, which in turn affects the quality of location based services negatively.

In this paper, we draw the attention of the spatial database community to the importance of road network compression while preserving the quality of map-matching. Our contributions can be summarized as follows:

- We present *COMA*, a lossy compression technique that significantly reduces the size of a given road map and that is sensitive to the quality of map-matching.
- Map-matching can be performed directly on the compressed data without the need to decompress the data beforehand.
- We relieve ourselves from the constraint that the original and compressed maps need to be visually similar. Hence, we aggressively achieve high compression ratios in areas where the map matcher is not confused by deformations in the map appearance that result from the lossy nature of the proposed technique.
- We introduce a tuning parameter, the *conflict factor*, that controls the behavior of the technique and trades the compression ratio for the map-matching quality.
- We provide an experimental study that uses real road maps and real GPS tracks to evaluate the performance of the proposed technique under variable GPS sampling rates, variable conflict factors, and variable levels of noise as measured in both urban and rural areas.

The rest of this paper is organized as follows. Section II highlights related work. Section III provides a formal definition of the problem. The proposed technique is described in Section IV. An experimental evaluation that is based on real road network data is given in Section V. Finally, Section VI concludes the paper.

II. Related Work

In this section, we overview road network compression techniques and we refer the reader to [10] for additional details. We categorize compression techniques in two main groups: (1) lossless compression and (2) lossy compression techniques. In lossless compression, every single data element is recovered when the given compressed map is decompressed back to its original format. Lossless compression is very important in terms of preserving the topological properties of a map. Alternatively, in lossy compression, certain spatial data is lost permanently as a result of the compression. Lossy compression is acceptable, or even desired, in cases where not all object details are required to perform the spatiotemporal operation in question.

Zongyu [17] proposes a lossless compression technique that navigates through the given road map based on its topology to build a prediction model. This model predicts the next to-be-visited node based on the already visited nodes. This compression scheme encodes a node using less number of bits than originally required. Suh et al. [7] propose another lossless approach that utilizes combinatorial optimization and data mining techniques to compress the road network nodes as well as the road shapes.

Lossy compression techniques, in general, discover similar chunks of data, create dictionaries on frequently referenced data chunks, and then refer to items in these dictionaries to encode the data. The higher the redundancy in the input data is, the higher the compression ratio is. Shashi et al. [15] propose a dictionary based compression technique, where the dictionary entries represent frequent shapes of line segments on the map. During data compression, line segments of similar shapes are extracted and represented by a single representative line segment. This representative line segment is inserted into the dictionary. Upon data decompression, the dictionary is looked up and decompression is done by reverting each line segment back to its representative line segment from the dictionary.

The reference line approach is another lossy compression approach that is proposed in [1], [3]. The basic steps of the algorithm can be described as follows: (1) For each polyline in the original map space, a reference line is identified, (usually produced from connecting the two ends of the polyline). (2) The coordinates of that reference line along with its angle from the original coordinate system is used to apply an affine transformation to the points on that polyline. (3) The delta distances in the vertical direction between the intermediate points on the polyline and the reference line in the new coordinate system are bounded by a predefined error threshold e. The selected reference line should keep these deltas within e, otherwise, a more representative reference line is selected. (4) In the aggressive mode of the reference line approach [1], which achieves higher compression ratio but less accurate decompression, the original coordinate values of the two ends of the line are stored, along with the number of intermediate points and the error tolerance e. In the less aggressive one [3], (less lossy and less compression ratio), the algorithm stores delta vectors between each intermediate point coordinates and the origin of the reference line, in addition to the two ends of the reference line themselves.

Map generalization is a process of reducing the complexity of the map without hampering the topological and structural features [12]. Generalization operators include simplification and smoothing. One of the most known line generalization and simplification technique is the *Douglas-Peucker* algorithm [4]. Shin ting et al. [16] utilize an improved *Douglas-Peucker* algorithm to avoid self-intersections for any specified tolerance. Saalfeld [14] uses a convex hull to efficiently detect and correct the topological inconsistencies of the polyline with itself and with other polyline characteristics. Ali et al. [9] propose a hybrid aggregation and compression technique and integrate it with the query processing pipeline of a road network database.

III. Problem Definition

In this paper, we address the road network compression problem such that the output is sensitive to the quality of the map-matching operation. In this section, we give a formal definition of the problem and describe the input and output of the proposed compression algorithm (Section III-A). Then, we describe the input and output of a typical map-matching algorithm (Section III-B). Note that this paper proposes a novel algorithm to generate a compressed road map that is usable by any map-matching technique. Hence, the choice of the map matcher is orthogonal to the proposed compression algorithm. We also define two measures of performance, the compression ratio CR and the map-matching $accuracy$.

A. Road network compression

Consider a road network graph $G(N, E)$, such that:

- N, is a set of nodes, where each node $n_i(lat, lon) \in N$ is defined by its latitude (lat) and longitude (lon), and
- E, is a set of edges, where each edge $e_{s,e}(n_s, n_e, w_{se})$ $\in$ E is defined by a start node n_s, an end node n_e, and a weight w_{se} that refers to the cost of traversing this edge, e.g., distance or travel time.

We assume that the given road network graph G is directed, where the travel direction over edge e is from the edge's start node to the end node (and is represented as $e : n_s \rightarrow n_e$). An *undirected* edge means that this edge is bi-directional (and is represented as $e : n_1 \leftrightarrow n_2$). For example, an undirected edge e that connects nodes n_1 and n_2 will be converted into two edges with the same weight, one edge $e_{1,2}$ from n_1 to n_2 and another edge $e_{2,1}$ from n_2 to n_1.

The following definitions formalize the problem and introduce several concepts that are used throughout the rest of the paper:

Definition 1: **Road network compression** generates a compressed version of the road network graph $G^{'}(N^{'}, E^{'})$ such that $N^{'} \subset N$ and $|E^{'}| < |E|$.

Definition 2: **Victimized node.** A victimized node is a node n_v such that $n_v \in N$ and $n_v \notin N^{'}$.

Definition 3: **Bridge edge.** if n_v is a victimized node that is connected to nodes n_i and n_j by edges $e_{i,v} \in E$ and $e_{v,j} \in E$, respectively, $\exists$ a *bridge edge* $e_{i,j}(n_i, n_j, w_{ij}) \in E^{'}$ to reconnect n_i and n_j such that $w_{ij} = w_{iv} + w_{vj}$.

The definitions above implies that the compression problem generates a compressed graph $G^{'}$ such that the number of nodes is reduced by victimizing several nodes from the original graph G. Consequently, the nodes in the resultant graph $G^{'}$ is a subset of the nodes in the original graph G (as described in Definition 1). If two nodes n_i and n_j are connected through an intermediate node n_v that is victimized during the compression process (Definition 2), n_i and n_j are reconnected through a *bridge edge* to maintain the connectivity of the compressed graph (Definition 3). Hence, eliminating a victim node n_v also compresses two adjacent edges into one edge, the bridge edge. Note that as more adjacent nodes are victimized, the bridge edge can substitute multiple consequent edges. The weight of the bridge edge becomes the sum of the weights of the edges it substitutes. By replacing multiple consequent edges by a single bridge edge, the number of edges in $G^{'}$ becomes less than the number of edges in G as indicated by $|E^{'}| < |E|$ in Definition 1.

Definition 4: **Compression Ratio.** $CR = 1 - |N^{'}|/|N|$

We define the compression ratio as the reduction in the number of nodes in the generated graph relative to the original graph. Other compression ratio measures may also consider the reduction in the number of edges. In our algorithm, the reduction in the total number of edges is linearly correlated with the reduction in the number of nodes. Hence, we consider the reduction in the number of nodes as our compression ratio measure.

B. Map-matching over compressed graphs

An object trajectory $Traj$ is a chronologically ordered set of object's timestamped locations. Each tiemstamped location is on the form of (object-id, timestamp, latitude, longitude). A map-matched trajectory appends an edge id e to each object's location to denote the road segment (or the edge in the graph) the object is believed to be travelling on at that timestamp. To assess the performance of map-matching using a compressed road graph $G^{'}$ relative to the original graph G, the object's trajectory is map-matched using both graphs.

Definition 5: **Accurate match.** If an object location is map-matched to edge e using the road network graph G and is map-matched to edge $e^{'}$ using the compressed version of the road network graph $G^{'}$, an *accurate match* is declared if $e = e^{'}$ or $e^{'}$ is a bridge edge that encompasses e as one of its compressed underlying edges.

We define the accuracy of map-matching given a road network compression techniques as the percentage of accurate matches relative to the entire trajectory length.

Definition 6: **Map-matching accuracy under compression.** $Accuracy = \frac{|Traj_{accurate}|}{|Traj|}$

IV. The Compression Technique

In this section, we describe our proposed *COMA* technique for road network compression for map-matching. We start by briefing the main idea of the proposed technique, then we go through the algorithm details, and finally, we give an example to further illustrate the steps of the algorithm.

Main Idea. The main idea of the proposed *COMA* technique is to reduce the number of nodes and edges in the given road network graph such that the deletion of a node/edge will not cause map-matching ambiguity. As described in Section III, multiple edges are compressed and represented by a single *bridge edge*. A smart compression algorithm optimizes for a minimal amount of false positives and false negatives. On one side, we make sure that the to-be-added bridge edge is closer

Algorithm 1 *COMA*: Road Network Compression For Map-Matching

Input: Road Network Graph $G(N, E)$,
Conflict Factor Threshold $\mathcal{C}$

1: $\#Original\_Nodes \leftarrow$ Count(N)
2: **for each** node $n \in N$ **do**
3: /* Step 1: Select Candidate Victim Node*/
4: **if** Select_Candidate_Victim(G, n) **then**
5: $E_{in} \leftarrow$ set of input edges to n
6: $E_{out} \leftarrow$ set of output edge from n
7: /* Step 2: Check Conflict Edges*/
8: **Check_Conflict**(G, n, E_{in}, E_{out}, $\mathcal{C}$)
9: /* Step 3: Victimize Chosen Node*/
10: **Delete_And_Merge**(G, n, E_{in}, E_{out})
11: **end if**
12: **end for**
13: $\#Compressed\_Nodes \leftarrow$ Count(N)
14: CR = 1- $\frac{\#Compressed\_Nodes}{\#Original\_Nodes}$ // Compression Ratio
15: **Return** G, CR

to the to-be-deleted victim node (and its edges) than any other existing edge in the vicinity. Hence, the object that is travelling on the to-be-deleted edge can still be map-matched correctly to the bridge edge with no ambiguity or confusion with other edges. Consequently, we avoid false negative, where the object is *not* map-matched to the bridge edge while it is supposed to. On another side, we make sure that the to-be-added bridge edge has no edges that are closer than the to-be-deleted edges. Hence, an object travelling on a nearby edge is *not* mistakenly map-matched to the bridge edge. Consequently, we avoid false positives, where the object is map-matched to the bridge edge while it is travelling on a different edge.

In other words, to decide whether a node n_v qualifies for victimization or not, *COMA* examines the newly formed bridge edge $e_{i,j}(n_i, n_j)$, (resulted from connecting the two far ends, n_i and n_j of the input and output edges of n_v). If (1) the bridge edge is closer to the in-hand node n_v than any other edge in the vicinity and (2) if the to-be-deleted edges are the closest to the bridge edge, the node n_v is victimized and the new bridge edge replaces the edges of n_v in the graph.

To control the behavior of the compression algorithm, we define a tuning parameter, called the *conflict factor threshold* $\mathcal{C}$. The conflict factor of a candidate victim node n_v is the distance from the this node n_v to the to-be-added bridging edge relative the distance from n_v to the nearest edge in the vicinity. If the conflict factor of node n_v is below the specified conflict factor threshold $\mathcal{C}$, the victimization may take place. Otherwise, the victimization stops and no compression is achieved at that node. By leveraging $\mathcal{C}$, we can control the trade-off between the compression ratio and the map-matching quality. The higher $\mathcal{C}$ is, the higher the compression ratio we get, and the less the quality of map-matching we guarantee, and vise versa.

Algorithm. The pseudo code of the proposed compression technique is given in Algorithm 1. The algorithm takes as input the original road network graph G, and the conflict factor $\mathcal{C}$. As output, the algorithm returns the compressed version of the road network graph, and the compression ratio. The algorithm has three main steps that are described as follows.

Step 1: Select Candidate Victim Node. The compression process start from any arbitrary node in the underlying road network graph, (Line 3). Once we pick up a node, the algorithm examines the ability to delete (or victimize) this node from the graph. Yet, the algorithm applies some checks to make sure that the deletion of this node is safe from a graph connectivity perspective. This is done by calling the Select_Candidate_Victim(G, n) function which considers the in-hand candidate node n as a valid victim for deletion when any of the following conditions is valid.

(1) **Intermediate node.** n is an *intermediate node* if it is connected to only two different nodes, e.g., n_i, and n_j and $n_i \neq n \neq n_j$, and satisfies one of the following two cases.

- Case1: **Intermediate node of a one-directional path**. n has one input edge coming from n_i, and an output edge going to n_j, i.e., $n_i \rightarrow n \rightarrow n_j$.
- Case2: **Intermediate node of a bi-directional path**. the two nodes n_i, and n_j are connected to n via bi-directional edges, i.e., $n_i \leftrightarrow n \leftrightarrow n_j$.

(2) **Fan in/out node.** n is a *fan in* or *fan out* node if it is connected to more than two other nodes with one-directional edges, and there is only one input edge and all the remaining edges are output edges. Alternatively, there is only one output edge and all the remaining edges are input edges.

Intermediate nodes (both one-directional and bi-directional cases) are appealing for compression. Intermediate nodes can be victimized with minimal impact on the graph connectivity by simply *bridging* the victim node, i.e., connecting the nodes before and after the victim node by a bridge edge. Also, the fan-out nodes are bridged by connecting the start node of the input edge to the end nodes of all output edges directly. An example is detailed later in this section.

After we discussed the various cases where a node is considered for victimization, we highlight cases where a node is *never* considered for victimization.

- **Cornerstone node.** A cornerstone node has edges that either *all* input edges or *all* output edges. The deletion of such a node breaks the connectivity and/or directional flow of the graph.
- **Highly-connected node.** If a node n has multiple input edges and multiple output edges, the consequences of deleting this node will produce a large number of bridge edges to cover all connectivity possibilities. For example, if a node has x number of input edges and y number of output edges (i.e., a total of $x + y$ edges), deleting this node will result in $x \times y$ number of edges to reconnect all broken connection between the input edge sources and the output edge destinations.
- **Variable-directionality node.** If a node n has a mix of one-directional and bi-directional edges, the consequences of deleting this node will produce parts of the graph that violate the directional flow of the graph, i.e., the path between n_3 and n_5 is half one-directional and half bi-directional.

We deliberately exclude corner stone, multi-edge and variable directionality nodes from being victimization candidates in the algorithm.

Algorithm 2 `Check_Conflict Function`

Input: Road Network Graph $G(N, E, W)$, Node n, InEdges E_{in}, OutEdges E_{out}, Conflict Factor $\mathcal{C}$

1: **for each** edge $e_{in} \in E_{in}$ **do**
2: **for each** edge $e_{out} \in E_{out}$ **do**
3: $e_{conflict} \leftarrow$ Find nearest edge to n where $e_{conflict}$ is not connected to n
4: $e_{bridge} \leftarrow$ Create new edge by connecting the far ends of e_{in} and e_{out}
5: **if** Distance(n, e_{bridge}) / Distance($n, e_{conflict}$) $< \mathcal{C}$ **then**
6: $n_{mid} \leftarrow$ Get midpoint of e_{new}
7: $e_{newConflict} \leftarrow$ Find nearest edge to n_{mid} where $e_{newConflict}$ is not connected to n
8: **if** $e_{newConflict} = e_{conflict}$ **OR** Distance(n, e_{new}) / Distance($n, e_{newConflict}$) $< \mathcal{C}$ **then**
9: Mark $< n, e_{in}, e_{out} >$ as eligible victims
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **Return**

Algorithm 3 `Delete_And_Merge Function`

Input: Road Network Graph $G(N, E, W)$, Node n, InEdges E_{in}, OutEdges E_{out}

if All combinations of $\{< e_{in}, e_{out} >\} \in \{E_{in} \times E_{out}\}$ are marked for deletion **then**
for each $< e_{in}, e_{out} > \in \{E_{in} \times E_{out}\}$ **do**
$W(e_{bridge}) = W(e_{in}) + W(e_{out})$
Add e_{bridge} to G
end for
Delete n from G
Delete e_{in} and e_{out} from G
end if
Return

Step 2: Check Conflict Edges. For a selected candidate node n, our objective is to victimize this node and to replace each of its connected pairs of input/output edges $< e_{in}, e_{out} >$ with a single new bridge edge e_{bridge} that links the two far ends of that pair. However, before we victimize the node n, we check if the to-be-added bridge edge has enough distance away from nearby edges. This step makes sure that this compression is safe from a map-matching perspective. The pseudo code for the $check\_conflict$ function is given in Algorithm 2. The conflict check has two phases. The first phase of the conflict check considers the edges that are close to the candidate victim node n while the second phase considers edges that are close to the to-be-added e_{bridge}.

In the first phase, it finds out the closest edge $e_{conflict}$ to the in-hand node n, (Line 3 in Algorithm 2). After that, we create a new edge e_{bridge} by linking the start node of the input edge e_{in} and the end node of the output edge e_{out} of the under processing pair of edges $< e_{in}, e_{out} >$ around n,(Line 4). Next, (Lines 5 to 11 in Algorithm 2), we get the ratio between the distance from n to the bridge edge e_{bridge}, and the distance from n to the conflict edge $e_{conflict}$. If this ratio is less than the controllable parameter $\mathcal{C}$, the conflict factor threshold, e_{bridge} is far from nearby conflicting edges and, hence, may substitute the edge pair $< e_{in}, e_{out} >$ and avoid false negatives (as described above).

To avoid false positives and further map-matching conflicts, the second phase of the conflict check considers all edges in the vicinity of e_{bridge}. Among these edges, we find out the edge with the minimum perpendicular distance to the midpoint of e_{bridge} and we call it $e_{newConflict}$. If $e_{newConflict}$ refers the same edge of $e_{conflict}$, we conclude that the closest edge to the to-be-added edge e_{bridge} is the same the closest edge to the to-be-deleted node n_v. Hence, we mark the pair $< e_{in}, e_{out} >$ as safe to be deleted and replaced by the new edge e_{bridge}. if $e_{newConflict} \neq e_{conflict}$, we check how much $e_{newConflict}$ is of conflict relative to neighboring edges based on the specified conflict factor threshold $\mathcal{C}$. If the conflict of $e_{newConflict}$ is less than $\mathcal{C}$, we mark the pair $< e_{in}, e_{out} >$ as safe for deletion. Otherwise, we do not victimize the node or any of its edge and we move on to the following node in the graph.

Step 3: Victimize Node. The objective of this step is to perform two things, (1) deleting the victim node and its connected edges, and (2) adding the new bridge edge(s) to the graph. This is accomplished by calling the $Delete\_And\_Merge$ function, Algorithm 3. Initially, this function makes sure that all combinations of edge pairs $< e_{in}, e_{out} >$ in the set of input edges E_{in} and output edges E_{out} have passed the conflict check done in step 2. If this is the case, the algorithm proceeds by computing the weight for each new edge e_{bridge} by summing up the weights of its corresponding edge-pair $< e_{in}, e_{out} >$. Finally, e_{bridge} is inserted to the graph and the node n is eliminated. Consequently, the deletion of n triggers the elimination of its linked in and out edges from the graph.

At the end, after we visit all nodes and edges in the original graph, the algorithm computes the compression ratio to indicate how many nodes have been successfully removed from the graph based on the selected conflict factor threshold $\mathcal{C}$.

V. Experimental Evaluation

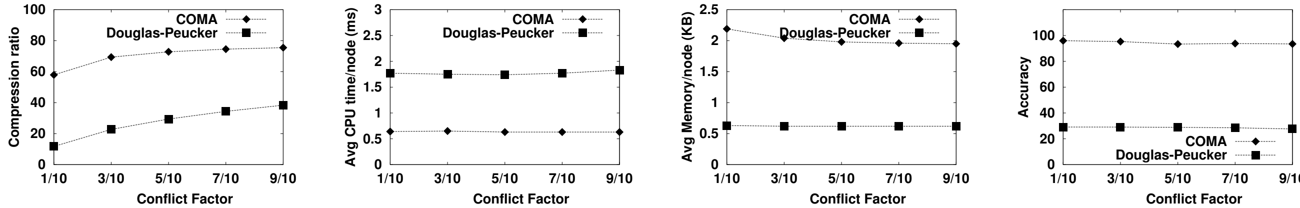
In this section, we evaluate the performance of our proposed *COMA* technique for compressing road networks while preserving the map-matching quality. We begin by describing the environment of the experiments. Next, we examine the effect of the *conflict factor* $\mathcal{C}$ on the compression ratio we can obtain as well as the performance measurements, i.e., CPU time and memory overhead. We use the *Douglas-Peucker* [4] algorithm as the competitive technique to our proposed *COMA* technique.

A. Experimental Setup

In all experiments of this evaluation, we use real road network graph of the Washington state, USA.

For the accuracy evaluation for the map-matching operation, we use real data sets for cars trajectories around the area of Seattle [2], [8]. In addition, we employ the Minnesota traffic generator [11] to generate larger sets of synthetic moving objects on the Washington road network.

All experiments are based on an actual implementation of the *COMA* and the competitive technique. All the components are implemented in C# inside visual studio 2013 with .net framework. All evaluations are conducted on a PC with Intel Xeon E5-1607 v2 processor and 32GB RAM, and running Windows 7.



(a) Compression Gain (b) CPU Efficiency (c) Memory Efficiency (d) Map-Matching Accuracy

Fig. 1. Effect of Conflict Factor on *COMA* VS *Douglas-Peucker*

B. *Efficiency Evaluation*

Initially, we study the influence of using different values for the *conflict factor* $\mathcal{C}$ on the compression ratio we can gain. We run both algorithms on the whole Washington graph. As given in Figure 1(a), we vary $\mathcal{C}$ from 0.1 to 0.9 on the x-axis and we measure the compression ratio we obtain on the y-axis. Obviously, the *COMA* technique achieves high compression ratio that starts at about 60% when $\mathcal{C}$ is 0.1 and keeps increasing until it reaches about 75% at $\mathcal{C}$ is 0.9. On the other side, the *Douglas-Peucker* achieves about 12% compression ratio at $\mathcal{C}$ = 0.1 and 38% at $\mathcal{C}$ = 0.9. These results prove that *COMA* outperforms the *Douglas-Peucker* in terms of compression ratio. It is also observed that both techniques achieve higher compression with larger $\mathcal{C}$ values, and vise versa. Figures 1(b), and 1(c) studies the efficiency of both techniques for the whole Washington state graph. This gives the average cost estimates for both CPU and memory overhead. It seems that both techniques have a steady trend in terms of CPU and memory costs. However, *COMA* is a CPU friendly technique whereas *Douglas-Peucker* is clearly a memory friendly technique. As given in Figure 1(d), *COMA* achieves high accurate map-matching that ranges from 96% at $\mathcal{C}$ = 0.1 with about 58% as compression ratio, to about 93.5% at $\mathcal{C}$ = 0.9 with compression ratio around 75%.

VI. Conclusion

While road network compression has been an active research problem, compression techniques aimed at high compression ratios regardless of the operations that are expected to be performed on the compressed version of the road map are the next generation of challenges that need to be addressed. We advance the state of the art along one such aspect: a compression technique to generate road network graphs that are consumable by the map-matching operations. Our proposed technique achieves high compression-ratios that reach up to 75% of the size of the original road network data while maintaining a high map-matching accuracy.

References

[1] Alexander Akimov, Alexander Kolesnikov, and Pasi Franti. Reference line approach for vector data compression. In *ICIP*, pages 1891–1894, Singapore, October 2004.

[2] Mohamed Ali, John Krumm, and Ankur Teredesai. ACM SIGSPATIAL GIS Cup 2012. In *ACM SIGSPATIAL GIS*, pages 597–600, California, USA, November 2012.

[3] Minjie Chen, Mantao Xu, and Pasi Frnti. Fast dynamic quantization algorithm for vector map compression. In *ICIP*, 2010.

[4] David H. Douglas and Thomas K. Peuker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973.

[5] Abdeltawab Hendawi, Eugene Sturm, Dev Oliver, and Shashi Shekhar. CrowdPath: A Framework for Next Generation Routing Services using Volunteered Geographic Information. In *SSTD*, Munich, Germany, August 2013.

[6] Abdeltawab M. Hendawi and Mohamed F. Mokbel. Panda: A Predictive Spatio-Temporal Query Processor. In *GIS*, California, USA, November 2012.

[7] Suh Jonghyun, Jung Sungwon, Pfeifle Martin, Vo Khoa T, Oswald Marcus, and Reinelt Gerhard. Compression of digital road networks. In *SSTD*, pages 423–440, Massachusetts, USA, July 2007.

[8] JOSM. An extensible editor for OpenStreetMap (OSM). http://josm.openstreetmap.de/wiki, January 2014.

[9] Ali Khoshgozaran, Ali Khodaei, Mehdi Sharifzadeh, and Cyrus Shahabi. A hybrid aggregation and compression technique for road network databases. *Knowledge and Information Systems*, 17(3):265–286, 2008.

[10] Amruta Khot, Abdeltawab Hendawi, Raj Katti, Anderson Nascimento, Ankur Teredesai, and Mohamed Ali. Road network compression techniques in spatiotemporal embedded systems: A survey. In *the International ACM SIGSPATIAL Workshop on Geostreaming, IWGS*, Dallas, TX, USA, November 2014.

[11] Mohamed F. Mokbel, Louai Alarabi, Jie Bao, Ahmed Eldawy, Amr Magdy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. MNTG: An Extensible Web-based Traffic Generator. In *SSTD*, pages 38–55, Munich, Germany, August 2013.

[12] Nabil H. Mustafa, Shankar Krishnan, Gokul Varadhan, and Suresh Venkatasubramanian. Dynamic simplification and visualization of large maps. *International Journal of Geographical Information Science*, 20(3):273–302, 2006.

[13] Mohammed A. Quddus, Washington Y. Ochieng, and Robert B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C-emerging Technologies*, 15(5):312–328, 2007.

[14] Alan Saalfeld. Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm. *Cartography and Geographic Information Science*, 26(1):7–18, 1999.

[15] Shashi Shekhar, Yan Huang, Judy Djugash, and Changqing Zhou. Vector map compression: a clustering approach. In *GIS*, pages 74–80, VA, USA, November 2002.

[16] Shin ting Wu and Mercedes Roco Gonzales Mrquez. A Non-Self-Intersection Douglas-Peucker Algorithm. In *SIBGRAPI*, pages 60–66, Ouro Preto, Brazil, August 2003.

[17] Zongyu Zhang. Vector road network compression : a prediction approach. In *ASPRS*, Reno, Nevada, USA, May 2006.