2020

Road network simplification for location-based services

Abdeltawab Hendawi
*University of Rhode Island*, hendawi@uri.edu

John A. Stankovic
*University of Virginia*, stankovic@cs.virginia.edu

Ayman Taha
*Technological University Dublin*, ayman.farahat@tudublin.ie

*See next page for additional authors*

Follow this and additional works at: [https://arrow.tudublin.ie/scschcomart](https://arrow.tudublin.ie/scschcomart)

Part of the Computer Sciences Commons

**Recommended Citation**
Hendawi, A., Stankovic, J.A., Taha, A. et al. Road network simplification for location-based services. Geoinformatica 24, 801–826 (2020). DOI: 10.1007/s10707-020-00406-x

This Article is brought to you for free and open access by the School of Computer Sciences at ARROW@TU Dublin. It has been accepted for inclusion in Articles by an authorized administrator of ARROW@TU Dublin. For more information, please contact arrow.admin@tudublin.ie, aisling.coyne@tudublin.ie, gerard.connolly@tudublin.ie, vera.kilshaw@tudublin.ie.

Funder: EU; Spanish Ministry of Science, Innovation and Universities; Galician Ministry of Education
Authors
Abdeltawab Hendawi, John A. Stankovic, Ayman Taha, Shaker El-Sappagh, Amr A. Ahmadain, and Mohamed Ali

This article is available at ARROW@TU Dublin: https://arrow.tudublin.ie/scschcomart/116
Road network simplification for location-based services

Abdeltawab Hendawi1,2 · John A. Stankovic3 · Ayman Taha2,4 · Shaker El-Sappagh5 · Amr A. Ahmadain6 · Mohamed Ali7

Received: 15 February 2018 / Revised: 16 April 2019 / Accepted: 2 April 2020 /
© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract

Road-network data compression or simplification reduces the size of the network to occupy less storage with the aim to fit small form-factor routing devices, mobile devices, or embedded systems. Simplification (a) reduces the storage cost of memory and disks, and (b) reduces the I/O and communication overhead. There are several road network compression techniques proposed in the 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 operation which is an essential component for all location-aware services. Map-matching matches a measured latitude and longitude of an object to an edge in the road network graph. In this paper, we propose a novel simplification technique, named COMA, that (1) significantly reduces the size of a given road network graph, (2) achieves high map-matching quality on the simplified graph, and (3) enables the generated compressed road network graph to be used directly in map-matching and location-based applications without a need to decompress it beforehand. COMA smartly deletes those nodes and edges that will not affect the graph connectivity nor causing much of ambiguity in the map-matching of objects’ location. COMA employs a controllable parameter; termed a conflict factor $C$, whereby location aware services can trade the compression gain with map-matching accuracy at varying granularity. We show that the time complexity of our COMA algorithm is $O(|N|\log|N|)$. Intensive experimental evaluation based on a real implementation and data demonstrates that COMA can achieve about a 75% compression-ratio while preserving high map-matching quality.

Road Network, Simplification, Compression, Spatial, Location, Performance, Accuracy, Efficiency, Scalability.

Keywords Spatial road network · Compression · Performance · Efficiency · Accuracy · Scalability

Abdeltawab Hendawi
hendawi@uri.edu; a.hendawi@fci-cu.edu.eg

Extended author information available on the last page of the article.
1 Introduction

Extensive availability of GPS-enabled devices has increased the need for routing and navigation services. The storage, processing, and transmission of road-network data are the biggest performance issues facing such services and is an important data management challenge. A 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 mobile 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 [28]. Map-matching is crucial for location aware services that answer queries based on the current [9, 22] and/or future objects’ location [10, 11, 14]. 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 simplified version of the road network data is required.

Map compression or simplification enables small size devices, e.g., smart watches, to carry the road map for large areas. The words compression and simplification are used interchangeably throughout the paper to mean converting the underlying road network to a simplified graph.

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 Drives (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. As forecasted by ABIresearch [31], the smart watches market is expected to range from 30 to 50 million watches per year. This expectation shows the importance of having a compact version of the road map that fits into tiny GPS-enabled devices such as smart watches.

Motivated by the above reasons, road-network simplification becomes an essential goal for spatial database researchers. In fact, there are several compression techniques proposed in the literature [1, 15, 17, 30, 34]. 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 simplified 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 and hence the location-based service.

We present a novel road network graph simplification technique, named COMA, that can significantly reduce the size of a given road-network graph without losing any critical information that might prevent map-matching process on the compacted road-network map. COMA users are mainly the developers of location-aware systems who need to include light weight version of a road network graph in their mobile applications. One of the COMA’s merits is that spatial operations such as Map-matching can be performed directly on the compressed data without the need to decompress the data beforehand. Also, the underlying map-matching algorithm does not need to be changed to work on the simplified graph.

In addition, COMA can 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. Furthermore, a tuning parameter, named the conflict factor, is introduced to control the behavior of the technique and trades the compression ratio for the quality (details) of the compressed map. Moreover, a rigid analysis on the computational cost of the COMA algorithm is provided. Finally, an extensive experimental study is conducted using real data sets of road maps and GPS trajectories to evaluate the performance of the proposed technique in various settings.

In this article, we are improving the preliminary version of compression (simplification) framework, COMA, [12, 13]. Here, the proposed simplification framework is extended by: (a) speeding up the map simplification algorithm by pruning needless calculations, (b) describing the system architecture and its main data structures, (c) providing the pseudo code for the simplification algorithm with detailed explanation along with step-by-step illustrative example, (d) studying the effect of the conflict factor as controlling parameter on both the compression gain and map-matching quality over the compact data, and (e) conducting an extensive experimental evaluation and comparison with the competitive work.

The main idea of our proposed compression technique is to selectively get rid of as many nodes as possible from the original road network. Such selective node removal also ensures that the affected edges will not cause a conflict with other nearby edges when we do map-matching for object location. A distance threshold parameter, termed conflict factor $C$, determines the ratio of the distance between the original node and the closest conflict edge to the distance between the to-be-added edge (after removing a node) and the closest conflict edge. In this way setting the conflict factor controls the compression ratio while preserving the accuracy of map-matching on the compressed map data. The larger the value of $C$, the more compressed is the map, and lesser the accuracy of map-matching.

The rest of this paper is organized as follows. Section 2 highlights related work. Section 3 provides a formal definition of the problem. The system model is described in Section 4. The compression technique is described in Section 5. While Section 6 gives an illustrative example. Section 7 provides the cost analysis and more pruning decisions. The experimental evaluation that is based on real road network data is given in Section 8. Finally, Section 9 concludes the paper.

2 Related work

In this section, we overview related work in both map matching and road network compression
2.1 Map matching

Matching an individual GPS point in the object’s path to its nearest edge in the road map does not provide adequate accuracy. Due to the noisy nature of GPS signals, the nearest edge to the measured GPS point may not be the actual edge the object is travelling on. Consequently, map-matching on a point-by-point basis, where each point is map-matched individually and independently of other points, usually results in a disconnected path. In this resultant path, the moving object appears to be jumping across network edges that are not connected at all. Map matching techniques take into consideration several geometric and probabilistic factors to enhance the map matching accuracy.

In addition to the proximity of a GPS point to an edge in the graph, White et al. [33] present several map matching algorithms that take into consideration the similarity of the object’s heading information to the angle of the road. They have also enhanced the proposed map matching algorithm by considering the connectivity constraints of the road map. Greenfeld [7] proposes a geometric approach for map matching that assesses similarity between the object’s path and the road segments using distance and orientation. The Frechet distance has been used to match a curve in the object’s path to the corresponding road segments [4, 8]. Kim et al. [19] consider the distance of a GPS point to an edge, the similarity of the path to the road geometry and the continuity of the path.

Some techniques tackle the map matching problem probabilistically as a Hidden Markov Model (HMM) problem, where the object transitions from one road segment to another according to a transition probability matrix. HMM-based approaches factor the connectivity of the road segments into the transition probability matrix, consider multiple path hypotheses simultaneously, and choose the path with the maximum likelihood. For example, Lamb and Thiebaux [21] applies a Kalman filter followed by a HMM. Krumm et al. [20] use a HMM to balance the measurement noise and path probabilities. HMM proved to be efficient under low sampling rates and sparse data sets [26]. The ACM SIGSPATIAL organized a programming contest, the SIGSPATIAL Cup 2012, and invited participants to come up with efficient map matching algorithms. Around thirty map matching submissions were evaluated against accuracy and speed and the result is documented in [3]. In this paper, we utilize one of the winning algorithms [23] to evaluate the accuracy of map matching on both the original and compressed road network graphs. For a literature review on map matching, the reader is referred to [28].

2.2 Road network compression

In this section, we overview road network compression techniques and we refer the reader to [18] 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 [34] 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. [15] 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. [30] 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, 5]. 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 $e$.

At the decompression phase, the algorithm runs two equations to predict the intermediate points of the original curve. The first to restore points closest to the left side of the reference line and the second for the points closest to the right end point of the line. Initially, the two ends of the reference line are leveraged to recover the first point nearest to the left side, e.g., predicting coordinates of $n_3$ using $n_1$ and $n_2$ in Fig. 1a. Then, the two most recent restored points, (from left and right), are used to predict the next point, e.g., restoring $n_4$ using $n_3$ and $n_2$ in Fig. 1b.

In the less aggressive solution [5], (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 [25]. Generalization operators include simplification and smoothing. One of the most known line generalization and simplification technique is the Douglas-Peucker algorithm [6].

![Fig. 1: Decompression By Predicting The Original Nodes](image-url)
Shin Ting et al. [32] utilize an improved Douglas-Peucker algorithm to avoid self-intersections for any specified tolerance. Saalfeld [29] 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. [17] propose a hybrid aggregation and compression technique and integrate it with the query processing pipeline of a road network database.

Although the aforementioned compression techniques can achieve considerable levels of compression, however, none of them considers the quality of using the compressed road network version to perform spatial operations, e.g., map-matching.

Our work here differentiates itself from the existing work in that our road network compression (simplification) algorithm takes into consideration the accuracy of performing map-matching operation on the produced simplified road network graph.

3 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 3.1). Then, we describe the input and output of a typical map-matching algorithm (Section 3.2). 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.

3.1 Road network compression (simplification)

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:

Road network compression (simplification) generates a simplified version of the road network graph $G'(N', E')$ such that $N' \subset N$ and $|E'| < |E|$.

Victimized node. A victimized node is a node $n_v$ such that $n_v \in N$ and $n_v \notin N'$.

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, then there 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 (simplification) problem generates a simplified 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 3.1). If two nodes $n_i$ and $n_j$ are connected through an intermediate node $n_v$ that is victimized during the simplification process (Definition 3.1), $n_i$ and $n_j$ are reconnected through a bridge edge to maintain the connectivity of the compressed graph (Definition 3.1). Hence, eliminating a victim node $n_v$ also simplifies two adjacent edges into one edge, the bridge edge. Note that as more adjacent nodes are victimized, one bridge edge can represent multiple consequent edges. The weight of the bridge edge becomes the sum of the weights of the edges it represents. 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 3.1. Surely, not all nodes will be victimized. Certain types of nodes are never considered for victimization such as nodes with large number of connected edges, and nodes that have all output edges (focal start nodes) or all input edges (focal end nodes).

**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.

### 3.2 Map-matching over compressed graphs

An object trajectory $Traj$ is a chronologically ordered set of object’s time-stamped locations. Each time-stamped location is in the form of (object-id, time-stamp, 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.

**Accurate map-match.** If an object location is map-matched to an edge $e$ using the road network graph $G$ and is map-matched to an 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.

After determining the accurate map match, we define the accuracy of map-matching given a road network compression technique as the percentage of accurate map matches relative to the entire trajectory length normalized by the number of corresponding merged edges.

**Map-matching accuracy under compression.** $Accuracy = \frac{\sum_{i=1}^{T} accurate(Traj_i)}{|Traj_i|m}$, where $accurate (Traj_i)$ is the number of accurately map-matched locations in the trajectory $Traj_i$, $m$ is the number of corresponding merged edges per the bridge edge $e'$ and $|Traj_i|$ is the number of all locations in the the trajectory $Traj_i$. 
The COMA framework consists of a map based UI, compression and map-matching modules, and a spatial-index each acting in sync with user or system specified conflict factor parameters to facilitate compression with map-matching for object trajectories.

4 System model

The overall COMA framework and its two main components, namely the Compression Module, and the Map-Matching Module are shown in Fig. 2. Users interact with the system through its map-based web interface to submit compression and map-matching requests, and also to set the system settings. As mentioned earlier, COMA users are mainly developers. They can specify areas on the map to be compressed before being employed in their applications.

Once the COMA framework receives a compression request it sends it to the compression module which runs on the selected area on the map. The R-tree spatial index is augmented inside COMA to accelerate the retrieval of those parts of the road map that intersect with the given area of interest, Fig. 3. If the user selects to test the map-matching accuracy on the produced compact graph, the system launches the map-matching module which in turns accesses the spatial index to retrieve the collocated set of objects’ trajectories and tries to match them to their corresponding roads and compare the matching correctness against the already known results.

Fig. 2 The COMA framework consists of a map based UI, compression and map-matching modules, and a spatial-index each acting in sync with user or system specified conflict factor parameters to facilitate compression with map-matching for object trajectories.

Fig. 3 Indexing Road Network Using R-Tree Structure in COMA
5 The compression/simplification module

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 does not cause map-matching ambiguity. Here, we need emphasis that the employed map-matching algorithm in this work relies solely on the distance between the object location and the nearby edges. However, there are many other map-matching algorithms that rely on other factors (e.g., angles, direction etc.). As described in Section 3, multiple edges are simplified and represented by a single *bridge edge*. A smart simplification 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 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 negatives, 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 $e_{i,j}(n_i, n_j)$, (resulting 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* $C$. 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 to 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 $C$, the victimization may take place. Otherwise, the victimization stops and no compression is achieved at that node. By leveraging $C$, we can control the trade-off between the compression ratio and the map-matching quality. The higher $C$ is, the higher the compression ratio we get, and the less the quality of map-matching, and vice versa.

**Algorithm 2** Check_Conflict function.

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

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

**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 $C$. 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 starts from any arbitrary node in the underlying road network graph, (Line 3). Once we pick a node, the algorithm examines the ability to delete (or victimize) this node from the given road network graph, (i.e., whole graph or specific region). 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., $n_i \rightarrow n \rightarrow n_j$. For example, $n_2$ in Fig. 4a is an intermediate node in the one-directional path from $n_1$ to $n_3$.
   - 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., $n_i \leftrightarrow n \leftrightarrow n_j$. For example, $n_5$ in Fig. 4a is an intermediate node in the bi-directional path from $n_4$ to $n_6$.

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 edge, and there is only one input edge and all the remaining edges are output edges (e.g., $n_7$ in Fig. 4a is a fan-out node). 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.

**Algorithm 3 Delete And Merge function.**

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

1: if All combinations of $\langle e_{in}, e_{out} \rangle \in \{E_{in} \times E_{out}\}$ are marked for deletion then
2: \hspace{1em} for each $\langle e_{in}, e_{out} \rangle \in \{E_{in} \times E_{out}\}$ do
3: \hspace{2em} $W(e_{bridge}) = W(e_{in}) + W(e_{out})$
4: \hspace{2em} Add $e_{bridge}$ to $G$
5: \hspace{1em} end for
6: Delete $n$ from $G$
7: Delete $e_{in}$ and $e_{out}$ from $G$
8: end if
9: Return
After we discuss 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, e.g., node $n_1$ in Fig. 4a. 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, e.g., node $n_6$ in Figure 4a, 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, e.g., node $n_4$ in Figure 4a, 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 cornerstone, Highly-connected, and variable-directionality nodes from being victimization candidates in the algorithm.

**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 $C$, 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 $n$. 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 $C$. If the conflict of $e_{newConflict}$ is less than $C$, 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: Victimizenode

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 \(C\).

6 An illustrative example

Here, an illustrative example is used to explain how COMA is working. Figure 4 gives an example to show the steps of the proposed compression algorithm. In this example, the original road network consists of 13 nodes and 15 edges (Fig. 4a). Also, the conflict factor \(C\) is set to 0.5.

The compression process can start from any node in the given graph. We arbitrarily start from node \(n_1\). Unfortunately, we find that \(n_1\) has no input edges and two output edges, \(e_1\) and \(e_6\). Hence, \(n_1\) is a cornerstone node and is not a candidate node for victimization, thus, we skip to the next node \(n_2\). Because node \(n_2\) has exactly one input edge \(e_1\), and one output edge \(e_2\) (i.e., an intermediate node), \(n_2\) is marked as a candidate victim and there is a possibility that it will be deleted from the graph. Yet, we have to check the conflict between the new bridge edge (i.e., \(e(n_1, n_3)\) that connects the two far end nodes of edges going to or going out of \(n_2\)) and the set of nearby edges. To do so, we define a circular region centered at the node in-hand \(n_2\) and its radius is equal to the length of the longest edge connected to \(n_2\), as shown in Fig. 4b. We get a set of edges in the vicinity that intersect with this region. Then we find out the conflicting edge, that is the closest edge to \(n_2\) among these vicinity edges, \(e_7\) in this case. Next, we compute the conflict factor as the value of the distance from \(n_2\) to bridge edge \(e(n_1, n_3)\) divided by the distance from \(n_2\) to the conflicting edge \(e_7\), then, we compare this conflict factor value to the conflict factor threshold \(C\). Obviously, this ratio is less than \(C\). Since \(e_7\) is also the closest edge to the midpoint of \(e(n_1, n_3)\), \(n_2\) passes the two phases of the conflict check. Therefore, \(n_2\) is deleted from the original graph, and its two connected edges, \(e_1\) and \(e_2\), are replaced by one new edge \(e(n_1, n_3)\). The weight of \(e(n_1, n_3)\) is equal to the sum of weights on \(e_1\) and \(e_2\).

We continue the compression by moving on to \(n_3\). In Fig. 4c we successfully victimize \(n_3\) after passing the two phases of the conflict check. Note that \(e_7\) is the closest to the midpoint of the new edge \(e(n_1, n_4)\) and \(e_8\) is the closest to \(n_3\) itself. In the first phase of the conflict check, the conflict factor is computed as the distance from \(n_3\) to \(e(n_1, n_4)\) divided by the distance from \(n_3\) to \(e_8\). In the second phase, the conflict factor is computed as the distance from \(n_3\) to \(e(n_1, n_4)\) divided by the distance from \(n_3\) to \(e_7\).

Our attempt to delete \(n_4\) fails because one of the two connected edges is a bi-directional edge (\(e_4\)) and the other one is one-directional (\(e_3\)). This means \(n_4\) is a variable-directionality node and is, hence, not a candidate for victimization. The deletion of \(n_5\) is smoothly completed as the nearest conflict edge \(e_9\) is much farther than the new edge \(e(n_4, n_6)\) (Fig. 4d). Deletion of the node \(n_7\) is a compound step (Fig. 4e). As \(n_7\) has one input edge and three
output edges (i.e., a fan out edge), the deletion process acts as if there are three copies of \( n_7 \), one for each \(< \text{input}, \text{output} >\) pair of edges, i.e., \(< e_6, e_7 >\), \(< e_6, e_{10} >\), \(< e_6, e_{14} >\). We delete \( n_7 \) from the three pairs and replace each pair of \(< \text{input}, \text{output} >\) edges by a newly added bridge edge. Thus, \( n_7 \) is deleted along with its connected edges \( e_6, e_7, e_{14}, e_{10} \). Then, we inserted three new bridge edges, \( e_{19}, e_{20}, e_{21} \) (Fig. 4f).

The algorithm proceeds to delete the node \( n_8 \). As seen in Fig. 4f, the closest edge to \( n_8 \) is \( e_{20} \), while the closest edge to the midpoint of the new edge \( e(n_1, n_9) \) is \( e_{17} \). Hence, we apply two conflict checks one after the success of the other. The first check is for the distance from \( n_8 \) to \( e(n_1, n_9) \) divided by the distance from \( n_8 \) to \( e_{20} \), and the other one is for the distance from \( n_8 \) to \( e(n_1, n_9) \) divided by the distance from \( n_8 \) to \( e_{17} \). Fortunately, both ratios are less than \( C \), therefore, \( n_8 \) is eliminated from the graph followed by the removal of \( n_9 \) in another straightforward step. This sequence of node victimization resulted in connecting \( n_1 \) and \( n_6 \) through the added edge \( e_{23} \) (Fig. 4g).

The processing of \( n_{13} \) is similar to what we did previously with \( n_8 \), as shown in Fig. 4h. Then, our attempt to get rid of \( n_{12} \) fails because the conflict check with edge \( e_{23} \) fails. Finally, we are able to victimize \( n_{10} \) leaving the compressed version of the road network graph with 5 nodes out of the 13 nodes in the original one, Fig. 4i.

### 7 Cost analysis and design decision

#### 7.1 Cost analysis

We start this section by listing the two main graph properties Road Network Graphs (RN graphs):

1. Road network graphs are very sparse graphs since the number of in- and out-edges of any vertex are at most 5
2. Road network graphs are also simply directed almost planar since they usually contain very few bridges and tunnels. It is well known that any planar graph has at most \( 3|N| - 6 \) edges so that any planar graph are \((3, 6)\)—sparse graph.

**Notation**

1. \(|N|\): Number of nodes in the road network graph
2. \(|E|\): Number of edges in the road network graph

We now proceed to prove the following two lemmas.

**Lemma 3** The average-case complexity of the COMA algorithm is \( O(|N| \log |N|) \)

**Proof** In algorithm 1, Road Network Compression for Map Matching, the for loop in line 2 iterates over the total number of nodes \(|N|\) in the RN graph and therefore, it generates a growth factor of \(|N|\).

In algorithm 2, Check_Conflict Function, the two for loops in lines 1 and 2, each generate a constant factor \( k \), related to the number of in-and out-edges of each node, but since, by assumption, the RN graph is sparse, then the value of \( k \) is \( \ll \) than the number of nodes \(|N|\). The find neatest edge to \( n \) function in line 3 of the algorithm is currently implemented using an R-tree algorithm space partitioning index which has an average complexity [27]
of $O(|N| \log |N|)$, we conclude that the average-complexity of the COMA algorithm is $O(|N| \log |N|)$ as claimed.

**Lemma 4** The worst-case complexity of the COMA algorithm is $O(|N|^2)$

**Proof** Since the worst-case complexity of the R-tree query implementing our find nearest neighbor algorithm is known to grow linearly with the number of nodes $|N|$, and since the for loop implemented in algorithm 1, iterates over all $|N|$, we can directly conclude that the worst-case complexity of the COMA algorithm is of $O(|N|^2)$.

7.2 Design decisions

In this section, two pruning rules are proposed to efficiently accelerate the COMA algorithm by getting rid of unnecessary computations.

**Lemma 1** Connected end nodes pruning: if the node $n$ is a candidate for victimization and the end nodes of the input/output edges $<e_{in}, e_{out}>$ are connected to each other, then $n$ can not be victimized.

**Proof** if $n$ is a candidate for victimization and the end nodes of its input/output edges are connected via another direct edge, then the to-be-added bridge edge will not be the closest edge to the to-be-deleted victim node, $n$. Additionally, the conflict factor of $n$ will be at least 1. As a result, the deletion $n$ will cause map-matching ambiguity and hence it will increase false negative. Consequently, the deletion of $n$ will be failed.

For example, the $n_{11}$ in Fig. 4i is a candidate for victimization because it has one input edge $e_{25}$, and one output edge $e_{12}$ but the edge $e_{24}$ connects the two end nodes $n_1$ and $n_{12}$, respectively. Thus, the removal of $n_{11}$ is rejected.

**Lemma 2** straight line pruning: if the node $n$ is candidate for victimization and the angle between its connected pairs of input/output edges $<e_{in}, e_{out}>$ is 180° then victimize $n$ without checking the conflict factor.

**Proof** if $n$ is candidate for victimization and the angle between its connected pairs of input/output edges $<e_{in}, e_{out}>$ is 180° then $n$ is located at the new bridge edge $e_{bridge}$ and the distance between $n$ and $e_{bridge}$ equals 0. Consequently, The conflict factor of $n$ is zero.

For example, $n_3$ in Fig. 4a is a candidate for victimization and $n_3$ is located at the bridge edge, $e(n_2, n_4)$. Therefore, $n_3$ should be removed from the compressed map without computing its conflict factor.

8 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. Then, we describe the competitive compression technique against which we compare the COMA technique. Next, we examine the effect of the
conflict factor $C$ on the compression ratio we can obtain as well as the performance measurements, i.e., CPU time and memory overhead. After that, we study the effect of different areas of the underlying graph on the behavior of the COMA technique. Finally, we test the map-matching quality of the resultant compressed graph.

8.1 Experimental setup

In all experiments of this evaluation, we use real road network graph of 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, 16]. In addition, we employ the Minnesota traffic generator [24] 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 Douglas-Peucker [6] as a competitive technique, Section 8.2. 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.

8.2 Competitive technique

We use the Douglas-Peucker [6] algorithm as the competitive technique to our proposed COMA technique. Douglas-Peucker is original introduced to reduce the number of points required to represent a given polyline. The reason for choosing this technique to compare with is that it can shrink the size of the road network graph, (when given as a set of polylines), at the same time, the produced compact graph still can be leveraged directly to perform map-matching operations. To make a fair comparison, we use the conflict factor $C$ as the distance threshold that is required by the Douglas-Peucker to guide its compression process. Here, we compute $C$ as the ratio between, the distance from a given polyline, (to-be-compressed edge(s) in the underlying road map), to the to-be-produced simplified edge, divided by the distance from that polyline to the nearest other conflict edge.

The main idea of the Passby algorithm is to consider the road intersections as the flag points at which the map-matching process focuses more. Once an object’s trajectory passes by an intersection, the algorithm finds out those edges around this intersection and select the one that are closer to more GPS points in the underlying trajectory. To achieve this, the algorithm takes two successive GPS points, the current point $p_{current}$ and its previous point $p_{previous}$, and computes a number of measurements for each nearby edge. It measures the projected distance between the edge and each of the two points and also the angle between the line connecting $p_{previous}$ and $p_{current}$, and the edge line. The GPS points will be linked to the edge with optimal measurements. In our map-matching test, we run the Passby algorithm on a set of objects trajectories $Traj$ for both the original road network graph $G$ and the compressed version $G'$. Then we measure how close the map-matching quality on the compact version to the original one.

8.3 Evaluation of compression gain

In this set of experiments, we examine the amount of compression we achieve using the proposed COMA technique. Also, we compare the results versus the ones we get from the Douglas-Peucker technique.
Effect of the conflict factor Initially, we study the influence of using different values for the conflict factor $C$ on the compression ratio we can gain. We run both algorithms on the whole Washington graph. As given in Fig. 5a, we vary $C$ 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 $C$ is 0.1 and keeps increasing until it reaches about 75% at $C$ is 0.9. On the other side, the Douglas-Peucker achieves about 12% compression ratio at $C = 0.1$ and 38% at $C = 0.9$. 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 $C$ values, and vice versa. As given in Fig. 6, the accuracy of Map-matching decreases with the increase of conflict factor, trajectory sparseness, or trajectory length for both algorithms.

Effect of the area type To examine how the COMA compression results are affected by the surrounding nature around the given road network graph, we select six different regions to represent area types around forests, down-town, high-way, lake, seaside, and rural.

Figure 9 compares the COMA compression ratio versus the Douglas-Peucker for each of the previously mentioned area types. Clearly, the percentage of size reduction is influenced by the type of the neighbourhood of the given road map.

For example, in the forest areas, COMA can achieve at least 64% and up to 81% compression ratio at $C$ equals 0.1 and 0.9 respectively. Also, in down-town, the gain we get by COMA drops down to 50% at $C = 0.1$ and to 69% at $C = 0.9$.

In all areas, COMA defeats Douglas-Peucker by large difference. The reason behind this variability in the obtained compression ratio is that the area type defines the shape of the road network graph. For example, roads in down-town have more intersections, branching, and higher density, (i.e., number of nodes per unit of area), than the highways, forest, and lakes. In turns, it is easier to delete nodes from the graph of forest area than the one for down-town area.
Effect of node degree Here, the degree of a node is the average number of edges connected to this node. The overall trend of COMA in Fig. 7a is to decrease the compression ratio when the node degree increases. Basically, that is because, the larger the degree the more conflict edges we might find, and consequently the less the ability to delete nodes from the original graph.

Effect of road network density We use the number of nodes divided by the size of the area as an indicator of how dense the given road network graph in different areas in Washington. For example, 73K means there are 73,000 nodes per mile square, i.e., lat/long degree, of the road network in this area. For the same reason mentioned with the node degree, the compression we gain by COMA goes down when road network density goes up, Fig. 8a.

8.4 Efficiency evaluation

Figure 5b and 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. Except for the first value for COMA in Fig. 9c), 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.

Figures 10 and 11 give the results of studying the efficiency behavior of the two techniques with different area types. As shown in the former figure, COMA significantly reduces the CPU time required to compress a road graph compared to the Douglas-Peucker. We can also notice the influence of the area type on the average CPU cost. For example, COMA costs about 2.8ms at $C = 0.1$ to process the graph of down-town area, Fig. 10d, while it costs less than half millisecond for lakes at the same $C$, Fig. 10c. Douglas-Peucker has similar
trend of reacting to area type effect, but, with much higher CPU costs, 8.6ms and 1.92ms respectively.

From the memory overhead efficiency perspective, COMA is the loser here. The reason for these efficiency patterns is that COMA converts the given road network graph into extended version where intermediate nodes are converted into regular nodes and edges. This step can not be done for the Douglas-Peucker, as it needs a long polyline. By doing so, COMA occupies more memory. Moreover, COMA processes node by node without visiting the same node twice which is not the case for recursive visiting in the Douglas-Peucker. Thus, COMA is more CPU friendly.

![Fig. 9 Effect of Area Type on Compression Ratio](image)

![Fig. 10 Effect of Area Type on Efficiency (CPU Time)](image)
When we examine the effect of node degree, Fig. 7b and c, and road network density, Fig. 8b and c, on the COMA efficiency measurements, we find that it costs more CPU with larger degrees and density and vice versa for memory overheads. The reason is that larger degree/density means more checks for edges conflict which means more CPU time. This also means the same data structures can serve more nodes per unit which decreases the total memory overhead.

8.5 Testing the map-matching quality

In this set of experiments, we examine the accuracy of correctly map-matching locations of moving objects trajectories on the compact road network graph, Definition 3.2 and 3.2. As mentioned earlier, we use sets of real and synthetic moving objects trajectories distributed over the road network graph of Washington, USA.

As given in Fig. 6a, COMA achieves acceptable accurate map-matching that ranges from 49.9% at $C = 0.1$ with about 58% as compression ratio, Fig. 5a, to about 35.8% at $C = 0.9$ with compression ratio around 75%. On the other side, Douglas-Peucker barely achieves 20.0% at $C = 0.9$ with compression ratio = 27.3% and its maximum accuracy comes at 27.4% when $C = 0.1$ with very low compression ratio = 11.7%.

In Fig. 6b, we check the effect of using different levels of trajectory sparseness on the map-matching quality. We vary the trajectory sampling from one point every 1 second to one point every 20 seconds. Generally, Douglas-Peucker is not sensitive to the trajectory sparseness, while COMA is negatively affected by sparse sampling rate. The reason is that COMA produces short edges, i.e., without intermediate nodes, which is not the case for Douglas-Peucker. Thus, skipping few seconds might jump the matching to the next edge and this does not give the Passby algorithm [23], a sufficient number of consecutive points on each single edge to do the right map-matching.

Figure 6c studies the effect of trajectory length on the map-matching accuracy. Both techniques have deceasing trend in the accuracy with longer trajectories. However, COMA loses less than 14% from its accuracy at length = 1min to 52% at length = 20min, while
Douglas-Peucker drops from 33.6% to 12.3% at length =1min and 20min respectively. A possible reason for that the Passby algorithm uses few trajectory points at the two ends of the vicinity edges to make map-matching decision. Once an edge is chosen, all points in-between those two ends will automatically be matched to that edge. If the decision is wrong, that will have larger negative effect on Douglas-Peucker accuracy than COMA because the former produce longer edges, have intermediate nodes.

Though in most cases COMA achieves an acceptable map-matching accuracy, For example, in forest area, Fig. 12e, the map-matching accuracy for COMA goes down from 47.11% to 30.38% at $C = 0.1$ and $0.9$ respectively. One reason for this is the nature of the forest environment, e.g., high dense trees, that badly affects the GPS accuracy. Hence, objects locations suffer from wider range of uncertain, yet, it is harder to be map-matched correctly to its correct edge.

### 8.6 Experiments summary

The conducted experiments prove the promises of COMA from three main perspectives. (1) From the compression achievements perspective, it can perform up to 75% compression ratio. (2) From the efficiency perspective, it is much faster than the Douglas-Peucker, as the main competitive technique. However, the later is more memory friendly than COMA. (3) From the map-matching accuracy perspective, COMA can be directed to get good accuracy, (i.e., by trying different $C$ values),and in general its accuracy does not go below 30% compared to 4% for the Douglas-Peucker technique.

### 9 Conclusion

In this paper, we highlight the importance of compressed road network maps from storage and communication perspective. With the proliferation of mobile, hand-held and embedded devices, the reduction in sizes of road maps becomes a metric that drives cost. 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 obtaining an acceptable map-matching accuracy. Experimental studies validate extensively the utility of our approach compared to existing techniques and are easily adaptable to existing device form-factors, the main aim of our work.

Acknowledgments We extend our thanks to Anas Basalamah (Umm Al-Qura University), Amruta Khot, Aqeel Rustum, and Ankur Teredesai (University of Washington Tacoma) for their comments and support in the initial phase of the work. This research is partially funded by the European Union’s Horizon 2020 research and innovation program under the Marie Skłodowska-Curie Grant Agreement No. 860621. It is also partially supported by the Spanish Ministry of Science, Innovation and Universities (grants RTI2018-099646-B-I00, TIN2017-84796-C2-1-R, TIN2017-90773-REDT, and RED2018-102641-T) and the Galician Ministry of Education, University and Professional Training (grants ED431F2018/02, ED431C2018/29, ED431G/08 and ED431G2019/04). Some of the previous grants were co-funded by the European Regional Development Fund (ERDF/FEDER program).

References

1. Akimov A, Kolesnikov A, Franti P (2004) Reference line approach for vector data compression. In: Proceeding of the IEEE international conference on image processing, ICIP, pp 1891–1894, Singapore
2. Ali M., Krumm J., Teredesai A (2012) ACM SIGSPATIAL GIS Cup 2012. In: Proceedings of the ACM SIGSPATIAL international conference on advances in geographic information systems, ACM SIGSPATIAL GIS, pp 597–600, California, USA
3. Ali M. H., Krumm J., Rautman T., A. Teredesai. (2012) ACM SIGSPATIAL GIS cup 2012. In: Proceedings of the ACM international conference on advances in geographic information systems. ACM GIS, pp 597–600
4. Brakatsoulas S., Pfoser D., Salas R., Wenk C. (2005) On map-matching vehicle tracking data. In: Proceedings of the international conference on very large data bases, VLDB, pp 853–864
5. Chen M., Xu M., Franti P. (2010) Fast dynamic quantization algorithm for vector map compression. In: Proceeding of the IEEE international conference on image processing, ICIP
6. Douglas D. H., Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The International Journal for Geographic Information and Geovisualization, Cartographica 10(2):112–122
7. Greenfeld JS (2002) Matching gps observations to locations on a digital map. In: the 81th annual meeting of the transportation research board, Washington, DC, USA
8. Alt H, Efrat A, Rote G, Wenk C (2003) Matching planar maps. J Algorithms 49:262–283
9. Hendawi A., Sturm E., Oliver D., Shekhar S. (2013) CrowdPath: A framework for next generation routing services using volunteered geographic information. In: Proceedings of the international symposium on advances in spatial and temporal databases, SSTD, Munich, Germany
10. Hendawi A. M., Bao J., Mokbel M. F. (2013) iRoad: A framework for scalable predictive query processing on road networks. In: Proceedings of the international conference on very large data bases, VLDB, Riva Del Garda, Italy
11. Hendawi A. M., Bao J., Mokbel M. F., Ali M. (2015) Predictive Tree: An efficient index for predictive queries on road networks. In: Proceedings of the international conference on data engineering, ICDE, Seoul, South Korea
12. Hendawi A. M., Khot A., Rustum A., Basalamah A., Teredesai A., Ali M. (2015) COMA: Road network compression for map-matching. In: Proceedings of the international conference on mobile data management, MDM, Pennsylvania USA
13. Hendawi AM, Khot A, Rustum A, Basalamah A, Teredesai A, Ali M (2015) A map-matching aware framework for road network compression. In: IEEE MDM, pp 307–310, Pittsburgh, Pennsylvania, USA
14. Hendawi A. M., Mokbel M. F. (2012) Panda: A predictive spatio-temporal query processor. In: Proceedings of the ACM international conference on advances in geographic information systems, ACM GIS, California USA
15. Jonghyun S, Sungwon J, Martin P, Marcus VKTO, Gerhard R (2007) Compression of digital road networks. In: Proceedings of the international symposium on advances in spatial and temporal databases, SSTD, pp 423–440. Massachusetts, USA
16. JOSM (2014) An extensible editor for OpenStreetMap (OSM). http://josm.openstreetmap.de/wiki
17. Khoshgozaran A., Khodaei A., ShariFzadeh M., Shahabi C. (2008) A hybrid aggregation and compression technique for road network databases. Knowl Inf Syst 17(3):265–286
18. Khot A, Hendawi A, Katti R., Nascimento A., Teredesai A., Ali M. (2014) Road network compression techniques in spatiotemporal embedded systems: A survey. In: the International ACM SIGSPATIAL workshop on geostreaming, IWGS, Dallas, TX, USA
19. Kim S., Kim J.-H. (2001) Adaptive fuzzy-network-based c-measure-matching algorithm for car navigation system. IEEE Trans Ind Electron 48(2):432–441
20. Krumm J., Letchner J., Horvitz E. (2007) Map matching with travel time constraints. In: Society of automotive engineers, SAE, Detroit, Michigan, USA
21. Lamb P., Thiebaux S. (1999) Avoiding explicit map-matching in vehicle location. In: the 6th world conference on intelligent transportation systems, ITS, Toronto, Canada
22. Li Y, George S, Apfelbeck C, Hendawi AM, Hazel D, Teredesai A, Ali M (2014) Routing service with real world severe weather. In: Proceedings of the ACM SIGSPATIAL international conference on advances in geographic information systems, ACM SIGSPATIAL GIS, Texas, USA
23. Liu K, Li Y, He F, Xu J, Ding Z (2012) Effective map-matching on the most simplified road network. In: Proceedings of the ACM international conference on advances in geographic information systems, ACM GIS, pp 609–612. Redondo Beach, CA, USA
24. Mokbel M. F., Alarabi L., Bao J., Eldawy A., Magdy A., Waytas E., Yackel S. (2013) MNTG: An extensible web-based traffic generator. In: Proceedings of the international symposium on advances in spatial and temporal databases, SSTD, pp 38–55. Munich, Germany
25. Mustafa N. H., Krishnan S., Varadhan G., Venkatasubramanian S. (2006) Dynamic simplification and visualization of large maps. Int J Geogr Inf Sci 20(3):273–302
26. Paul N, John K (2009) Hidden Markov map matching through noise and sparseness. In: Proceedings of the ACM international conference on advances in geographic information systems, ACM GIS, pp 336–343. Seattle, Washington
27. Qi J., Tao Y., Chang Y., Zhang R. (2018) Theoretically optimal and empirically efficient r-trees with strong parallelizability. Proc VLDB Endowment 11(5):621–634
28. Quddus M. A., Ochieng W. Y., Noland R. B. (2007) Current map-matching algorithms for transport applications: State-of-the-art and future research directions. Trans Res Part C-emerging Technol 15(5):312–328
29. Saalfeld A. (1999) Topologically consistent line simplification with the Douglas-Peucker algorithm. Cartogr Geogr Inf Sci 26(1):7–18
30. Shekhar S., Huang Y., Djugash J., Zhou C. (2002) Vector map compression: A clustering approach. In: Proceedings of the ACM international conference on advances in geographic information systems, ACM GIS, pp 74–80, VA USA
31. Spencer N. (2015) The apple watch and smart watch forecast for 2015. https://www.abiresearch.com/market-research/product/1021800-the-apple-watch-and-smart-watch-forecast.html
32. Ting Wu S, Marquez MRG (2003) A non-self-intersection Douglas-Peucker algorithm. In: Brazilian symposium on computer graphics and image processing, SIBGRAPI, pp 60–66, Ouro Preto, Brazil
33. White C. E., Bernstein D., Kornhauser A. L. (Dec. 2000) Some map matching algorithms for personal navigation assistants. Trans Res Part C., Emerging Technol 8(1-6):91–108
34. Zhang Z. (2006) Vector road network compression: A prediction approach. In: Proceedings of the American society for photogrammetry and remote sensing conference, ASPRS, Reno, Nevada USA

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.
Abdeltawab Hendawi is currently an Assistant Professor in the Department of Computer Science and Statistics at the University of Rhode Island, and the Co-director of the AI-Lab at URI. He obtained my MSc and PhD in Computer Science and Engineering from the University of Minnesota, Twin Cities. His research interests are in big data management and analytics with applications in smart city and smart health. Dr. Hendawi has won a number of awards, including the Best Paper Award at the ACM SIGSPATIAL MobiGIS 2012, the Best Design/Plan Poster Award at the U-Spatial Symposium 2013; Best Overall Poster Award, also at the U-Spatial Symposium 2013; the Best Demo Paper Award at the ACM SIGSPATIAL 2014; and Best and Second Best Demo Paper Awards at the IEEE Mobile Data Management (MDM) 2015. My vision for the future research challenges in the area of location-aware services also won a highly recognized Award at the ACM SIGSPATIAL 2016. In addition, he is the recipient of the Hobby Postdoctoral Research Fellowship 2015-2018 from the Computer Science Department at the University of Virginia, and the Best Poster Award at the UVa Research Symposium 2018; and Best Demo Paper Award Runner-up at ACM SIGSPATIAL 2018. He has been a co-chair for the IEEE Big Spatial Data workshop (BSD) 2016 to 2019.

John A. Stankovic is the BP America Professor in the Computer Science Department at the University of Virginia. He served as Chair of the department for 8 years. He is a Fellow of both the IEEE and the ACM. He has been awarded an Honorary Doctorate from the University of York. He won the IEEE Real-Time Systems Technical Committee’s Award for Outstanding Technical Contributions and Leadership. He also won the IEEE Technical Committee on Distributed Processing’s Distinguished Achievement Award (inaugural winner). He has seven Best Paper awards, including one for ACM SenSys 2006. He also has two Best Paper Runner Up Awards, including one for IPSN 2013. He has also been a finalist for multiple other Best Paper Awards. Stankovic has an h-index of 107 and over 43,000 citations. In 2015 he was awarded the Univ. of Virginia Distinguished Scientist Award, and in 2010 the School of Engineering’s Distinguished Faculty Award. He also received a Distinguished Faculty Award from the University of Massachusetts. He has given more than 40 Keynote talks at conferences and many Distinguished Lectures at major Universities. Currently, he serves on the National Academy’s Computer Science Telecommunications Board. He was the Editor-in-Chief for the IEEE Transactions on Distributed and Parallel Systems and was founder and co-editor-in-chief for the Real-Time Systems Journal. His research interests are in real-time systems, wireless sensor networks, wireless health, cyber physical systems, and the Internet of Things. Prof. Stankovic received his PhD from Brown University.

Ayman Taha is an assistant professor at faculty of computes and Information at Cairo University- Egypt. Currently, Ayman is a Marie Sklodowska- Curie Career-FIT postdoc research Fellow at the Technological University Dublin (TUD), Ireland. Ayman’s research interests are GIS Systems, Spatial Data Computing, Data Mining, Categorical Data Analysis, Outliers Detection, Anomalous Events Identification, Machine Learning and Deep Learning. He has got the best PhD thesis at Cairo University. Dr. Ayman obtained his MSC, and PhD from the Cairo University.
Shaker El-Sappagh received the bachelor degree in computer science from Information Systems Department, Faculty of Computers and Information, Cairo University, Egypt, in 1997, and the master degree from the same university in 2007. He received the Ph.D. degrees in computer science from Information Systems Department, Faculty of Computers and Information, Mansura University, Mansura, Egypt in 2015. In 2003, he joined the Department of Information Systems, Faculty of Computers and Information, Minia University, Egypt as a teaching assistant. Since June 2016, he has been with the Department of Information Systems, Faculty of computers and Information, Benha University as an assistant professor. Currently he is Post-Doctoral Fellow at UWB Wireless Communications Research Center in the Department of Information and Communication Engineering at Inha University, South Korea. He has publications in clinical decision support systems and semantic intelligence. His current research interests include machine learning, medical informatics, (fuzzy) ontology engineering, distributed and hybrid clinical decision support systems, semantic data modeling, fuzzy expert systems, and cloud computing. Dr. El-Sappagh is a reviewer in many journals, and he is very interested in the diseases diagnosis and treatment researches.

Amr A. Ahmadain received his Master’s degree in Computer Science from the University of Louisville in 2002 and his PhD degree from the University of Cincinnati in 2014. For his outstanding scholarly achievements, in 2004, Dr. Ahmadain was awarded the University of Cincinnati University Research Council (URC) fellowship and, in 2009, he was also awarded Mary. J. Hanna fellowship. Dr. Ahmadain is currently a Research Associate at the University of Virginia.

Mohamed Ali is an associate professor at the Institute of Technology, University of Washington, Tacoma. Mohamed’s research interests include the processing, analysis and visualization of data streams with geographic and spatial information. For the past decade, Mohamed has been building commercial spatiotemporal data streaming systems to cope with the emerging Big Data requirements. In 2006, Mohamed and his colleagues at the database group at Microsoft Research ramped up the Complex Event Detection and Response (CEDR) project. Then, Mohamed joined the SQL Server group at Microsoft to productize the CEDR project. CEDR has shipped and brand-named as Microsoft StreamInsight®. Since the first public release of StreamInsight, Mohamed has been advocating for “real-time spatiotemporal data management everywhere”; that is the use of StreamInsight in monitoring, managing and mining real time geospatial information across a diversity of verticals. These verticals include but are not limited to online advertising, behavioral targeting, business intelligence, computational finance, traffic management, social networking, homeland security, emergency and crisis management. In 2011, Mohamed started another journey at Microsoft Bing Maps where he became at the frontline with the Big Data challenge and where he battled various types of spatial search queries. In 2014, Mohamed joined the University of Washington, Tacoma where he leads the geospatial data science team at the Center for Data Science.
Affiliations

Abdeltawab Hendawi\textsuperscript{1,2} · John A. Stankovic\textsuperscript{3} · Ayman Taha\textsuperscript{2,4} · Shaker El-Sappagh\textsuperscript{5} · Amr A. Ahmadain\textsuperscript{6} · Mohamed Ali\textsuperscript{7}

John A. Stankovic
Stankovic@cs.virginia.edu

Ayman Taha
a.taha@fci-cu.edu.eg; ayman.farahat@tudublin.ie

Shaker El-Sappagh
shaker.elsappagh@usc.es

Amr A. Ahmadain
aaa9aj@virginia.edu

Mohamed Ali
mhali@uw.edu

\textsuperscript{1} Department of Computer Science, Department of Computer Science and Statistics, University of Rhode Island, Kingston, RI, USA
\textsuperscript{2} Faculty of Computers and Artificial Intelligence, Cairo University, Giza, Egypt
\textsuperscript{3} Department of Computer Science, University of Virginia - Charlottesville, 85 Engineer’s Way, Charlottesville, VA 22904-4740, USA
\textsuperscript{4} School of Computer Science, Technological University Dublin, Dublin, Ireland
\textsuperscript{5} Centro Singular de Investigacion en Tecnoloxias Intelixentes, Universidade de Santiago de Compostela, Santiago, Spain
\textsuperscript{6} University of Virginia - Charlottesville, 85 Engineer’s Way, Charlottesville, VA 22904-4740, USA
\textsuperscript{7} School of Engineering and Technology, University of Washington Tacoma, 1900 Commerce Street, Tacoma, WA 98402-3100, USA