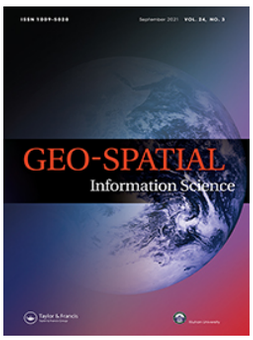

Geo-spatial Information Science

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tgsi20

An adaptive Markov chain algorithm applied over map-matching of vehicle trip GPS data

Bilge Kaan Karamete, Louai Adhami & Eli Glaser

To cite this article: Bilge Kaan Karamete, Louai Adhami & Eli Glaser (2021) An adaptive Markov chain algorithm applied over map-matching of vehicle trip GPS data, Geo-spatial Information Science, 24:3, 484-497, DOI: 10.1080/10095020.2020.1866956

To link to this article: https://doi.org/10.1080/10095020.2020.1866956

GEO-SPATIAL INFORMATION SCIENCE
2021, VOL. 24, NO. 3, 484–497
https://doi.org/10.1080/10095020.2020.1866956

Taylor & Francis
Taylor & Francis Group

An adaptive Markov chain algorithm applied over map-matching of vehicle trip GPS data

Bilge Kaan Karamete, Louai Adhami and Eli Glaser

Engineering, Geospatial, Kinetica DB Inc, Arlington, VA, USA

ABSTRACT

Markov chains have frequently been applied to match the probable routes with a set of GPS trip data that a pilot vehicle is emitting over a specific graph road network. This class of map-matching (MM) algorithms presently demonstrates and involve statistical and ad-hoc measures to drive the Markov chain transitional probabilities in picking the best route combinations constrained over the graph road network. In this study, we have devised an adaptive scheme to modify the Markov Chain (MC) kernel window as we move along the GPS samples to reduce the mistakes that can happen by the use of narrower MC widths. The measure for temporarily increasing the MC window width is chosen to be the ratio between the geodesic distance of current route to the actual geodesic distance between each pair of GPS samples. This adaptive use of MC has shown to have hardened the results significantly with tolerable computational cost increase. The details of the overall algorithm are depicted by the example routes extracted from various vehicle trips and the results are shown to validate the usefulness of the algorithm in practice.

ARTICLE HISTORY
Received 27 April 2020
Accepted 14 December 2020

KEYWORDS
Hidden Markov chain (HMC); map-matching (MM); graph networks

1. Introduction

The basic goal of a generic purpose map-matching algorithm is to find out the matching road segment (s) to a set of GPS data emitted from a traveling vehicle within some error of accuracy. There are many essential factors affecting GPS accuracy; the government provides the GPS signal in space with a global average User Range Rate error (URRE) of ≤ 0.006 m/s over any 3 second interval, with 95% probability. This measure must be combined with other factors outside the government's control, including satellite geometry, signal blockage, atmospheric conditions, and receiver design features/quality, to calculate a particular receiver's speed accuracy (US-Government 2020). Even though it is impossible to eliminate these measurement errors, a brute force approach of increasing the GPS sampling frequency is often used to minimize the cumulative errors at the expense of higher data accumulation and processing. The map-matching (MM) task is finding the most reasonable route corresponding to a number of GPS data samples under the assumption that the data has some unavoidable level of noise associated with it.

The road networks are generally conceptualized as unstructured topology composed of directed node-edge graphs $G\{V, E\}$ where every junction is modeled by a node V and each street by a road segment edge E emanating from V as its edges. The number of edges per node could be different and even though the road network graph is usually not modified often, the traffic conditions and loads mutate significantly during the course of daily traffic. Less frequently, new road segments to the graph could be added, deleted, or directions changed for restricted access. Therefore, a dynamic data structure for graph topology is required to model the road networks where there can be expanding or shrinking number of edges per node (Karamete et al. 2016; Boeing 2017; Xin et al. 2013). The cost (impedance) per edge is modeled as either the geodesic distance or as the time it takes between the two end vertices of the edge. If the speed (traffic allowed) is known at the road segment, the cost value can be calculated from the distance divided by the speed and associated with the edge. Various graph solvers make use of this scalar cost field to solve for the most optimal routing such as shortest path single routing (Dijkstra), multiple routing (traveling salesman), back-haul routing, map-matching, etc. (Kinetica 2020) (See Figure 1).

The main idea of MM in converging to a reasonable trip path is to minimize the error of propagation from one GPS point to the next while associating them to the supposed road segments. During this process, if an algorithmic error is made in associating a GPS sample to a wrong road segment, its propagation to the next sample point is unavoidable and the entire matching process results in an erroneous route selection. Hence, it is crucial to include a range of GPS samples in a broader sense instead of focusing on single sample points. The human perception is very apt to make more or less the "correct" routing decisions by looking at a broader

CONTACT Bilge Kaan Karamete kallespapaz@gmail.com

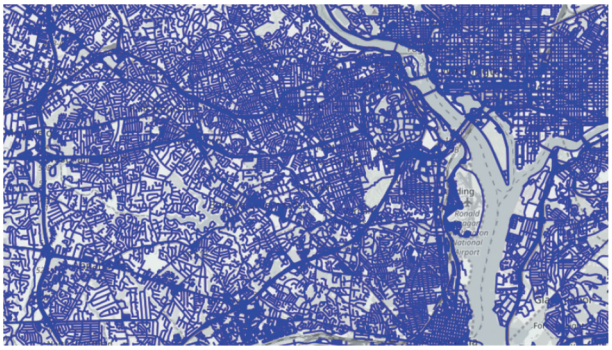

Figure 1. A typical road graph network – unstructured number of edges emanating from each node. Courtesy to HERE company – The graph of DC metropolitan area.

section of GPS samples than just a few. In other words, we are good at processing the visual information by looking ahead and corrections are almost instantly made among all possible paths around the sample points. Similarly, the inference mechanism of our minds in matching road segments to the GPS samples is unconsciously computing in a predictive-corrective manner by looking at a range of data to extract the most likely route as the shortest possible one by snapping GPS points to the nearest road segments. If these are the criteria of our innate inferencing process, then there are certainly the constraints of finding the shortest possible route and nearest road segments. In fact, there are many possibilities emerge when we consider the fuzzy (noise) aspect of the GPS data that the nearest snap location may not be the most "correct" road segment (Newson and Krumm 2009; Quddus, Ochieng, and Noland 2007). We can certainly find the best answer from the set of possible snap locations but we need to process possible path formations within a window of a number of GPS samples. In other words, we can not make a prediction for the "current" location without observing the GPS samples up ahead (future GPS point states).

The number of consecutive possibilities between each pair of GPS samples in a corrective manner should also take into account the constraints of the road network, i.e., some segments are only-one-way restricted, and the route can not include segments jumping off to another segment if there is no graph connections. There are also the constraints of plausibility, i.e., each sample has a time-stamp (breadcrumbs) and the trip sequencing should obey this ascending order for coherence (Newson and Krumm 2009; Goh et al. 2012). Moreover, there could be speed limits imposed over certain road segments and the trip path should not contradict the travelled distance, calculated between the speed limit and the timestamps of the GPS samples under the assumption that the vehicle actually obeys the traffic laws and regulations which is a reasonable assumption to make (except perhaps in Maryland and New Jersey). In light of these observations, a number of criteria for deducing the most likely route from the GPS samples can be stated as follows:

- Each GPS point should be snapping to one of its nearest road segments.
- The route should result in shortest paths among snapped locations calculated based on the weights (time or distance) of the road segments (Chen et al. 2014).
- The temporal order of the GPS points should be preserved in the routing.
- The route is constrained by the road network graph topology (connections).
- The route should obey the directed-ness of the segments, i.e., the edge directions of its graph topology.
- The travelled distance of the path should not contradict the speed limit of the road segments and cumulative timestamps.

To this end, above points are studied by various map-matching approaches devised from simple snapping to the nearest segments to the weighted averages of more sophisticated metrics brakat, (Brakatsoulas et al. 2005; White, Bernstein, and Kornhauser 2000). Most of these geometrical ideas often result in non-uniform level of success due to their reliance on sampling frequency and GPS accuracy. A relatively more successful strategy by (Brakatsoulas et al. 2005) used Fréchets distance between the curved approximation of the trace of the GPS samples to the road segments. Another approach was using the predicates that include sample heading and distance to the road angle and have a similarity measure leading to building of a topological measure where road constraints were also applied (Greenfeld 2002). These local techniques showed good results when sampling frequency and accuracy were adequate, however, they were also found to be inferior for lower sampling frequency cases. This was not a surprising finding as fewer noisy data is expected to result in poor routing matches. Lately, a rigorous comparative study is performed by Wei demonstrating various weight functions devised in the literature and their success rates using a Viterbi dynamic programming algorithm (Wei et al. 2012).

The procedure to solve this optimization problem should take into account the state of the GPS points in a range, i.e., if we are to figure out the projection of a GPS point onto a prospective set of segments at time t_n, the decision should not contradict for the next set of GPS points at t_{n+k} where k is the range or the width of the window that we "look ahead" to correct the predicted snap location at the current t_n station under the criteria and constraints listed above. The best-solving strategy for this kind of optimization problem is the application of Hidden Markov

Chains (HMC) concept where at each GPS station there is a probability computed from the transitional probabilities of the $+k$ ("ahead") states of the GPS points; in other words, the current state probability depends on the future possibilities.

This is how the MC is generally formulated for an MM problem (Newson and Krumm 2009): the hidden state is the likelihood (probability) of a GPS point to be snapped on a prospective edge segment. In other words, the probability of a point to be snapped over a prospective edge segment is hidden by the probabilities of the next set of GPS samples. These transitional probabilities could be modeled either as to how close the GPS point to a possible set of nearby edges or the cumulative cost of traveling from one prospective location to the next or both. To this end, there have been many MM algorithms tried in the literature using different approaches from totally topological to geometric and probabilistic and summarized by the survey paper of (Quddus, Ochieng, and Noland 2007).

The map-matching work of (Newson and Krumm 2009) utilizing Markov chains, exercised the geodesic distance between the consecutive GPS samples, z_i and z_{i+1} as being the base factor in determining the transitional probabilities in the HMC kernel. The deviation between this base distance and the route distance from road segment projections, r_i to r_j is defined as the raw transitional probability among all prospective snap projection combinations and depicted as t_{ij} in Equations 1(a-c). The distance error is then cast into an exponential probability distribution function shown in Equation 1(d) with β being an ad-hoc error coefficient, prior to the HMC kernel iterations depicted as k in Equations 1(e-f). The parameter definitions of Equation (1) are also illustrated in Figure 2. It is also worth noting the number of prospective snap locations (maximum of three, as yet another parameter) is only found within a preset radius of the R-tree (Guttman 1984) per GPS sample shown with the dotted circles in Figure 2. They have also noted that there can still be problems particularly for the noisy data near intersections even though HMC kernel superiorly predicts the ground truth with greater accuracy and not sensitive to the sampling frequency.

$$p_i = P(z_i|r_i) \tag{1a}$$

$$t_{ij} : p_i \mapsto p_j \tag{1b}$$

$$t_{ij} = \left| \left\| z_i - z_j \right\|_{geodesic} - \left\| r_i - r_j \right\| \right| \tag{1c}$$

$$t_{i,j} \leftarrow (1/\beta) e^{-t_{ij}/\beta} \tag{1d}$$

$$p_i^{k+1} = t_{ij} \cdot p_j^k \tag{1e}$$

$$p_i^{k+1} = t_{ij}^k \cdot p_j^0 \tag{1f}$$

Our algorithm borrows the main idea from Newson and Krumm but makes a major contribution in detecting the problem areas and adaptively applying wider HMC

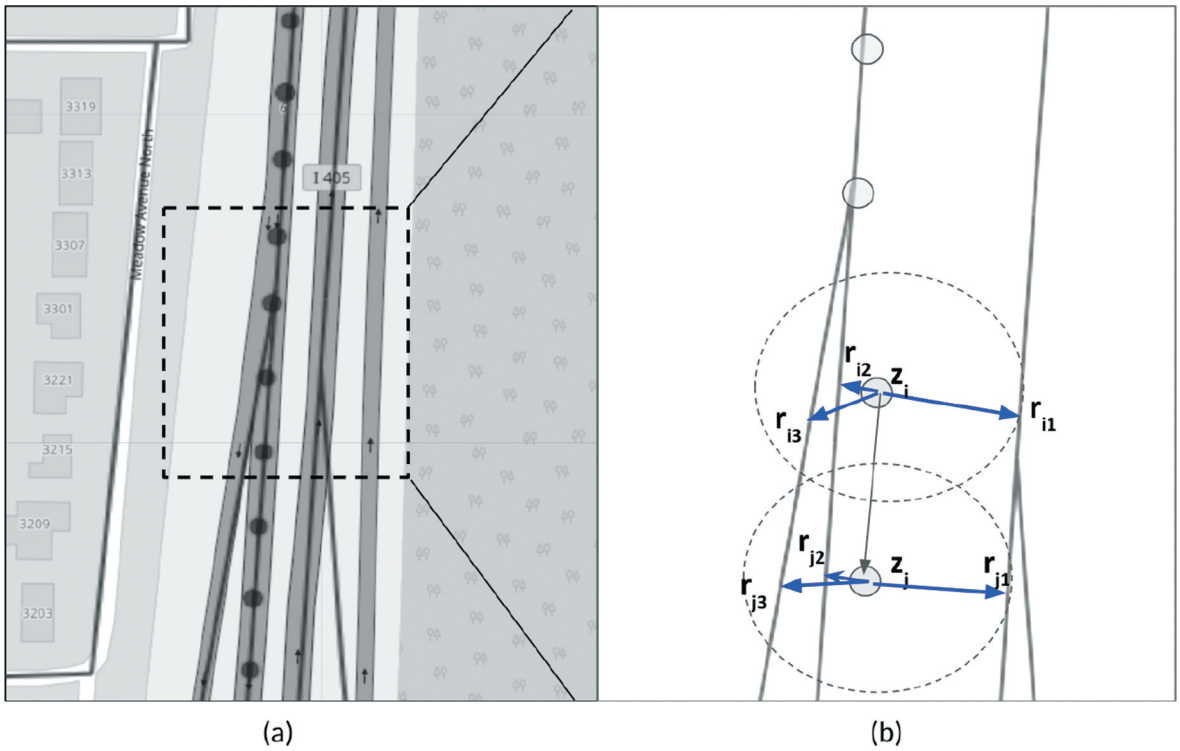

Figure 2. (a) A portion of GPS samples along a graph road network. (b) a sub-section of the samples where two consecutive GPS sample pairs and their respective parameters used in Equation (2), is shown. The parameters r_i and r_j are one of the three $(1-3)$ prospective snap locations for GPS samples z_i and z_j, respectively, found within the search radius of the R-tree shown with the dotted circles.

kernel widths where necessary to fix the remaining issues. The kernel width is then reset back to the shorter span for efficiency. The other major differentiating factor is that our process finds the total cost of the overall kernel sequence based on the aggregated sum of the mini Dijkstra runs within each pair of a sequence. Road weights are modified based on the relative distance of prospective snap locations of the road segments to the GPS points, so that the shortest path runs respect where all possible snaps would occur. Road constraints are applied as filters on the combination sequence generations before running mini-dijkstras on the probable paths. The overall algorithm is explained in the next Section 2, followed by its application on user test cases with varying sample frequencies and trip durations in Section 3.

2. Algorithm

The input to the MM algorithm is a set of time-stamped lon-lat pairs of GPS data and a road network graph. The road network graph $G(V, E)$ is generated enclosing the entire set of GPS samples and a range tree (R-Tree) is constructed from the line segments of the edges of the graph (Guttman 1984). A set of closest edge segments is then searched and cached for each GPS sample using the R-Tree. The number of prospective edges per GPS sample is a parameter of the algorithm and a default value of up-to three distinct edge segments is used within a search radius of 10 times the graph tolerance – the graph tolerance is usually chosen to be between $1 - 10$ mwith a corresponding lon-lat angle tolerance of roughly $10^{-5} - 10^{-4}$, respectively. The GPS samples are optionally filtered to remove noisy data due to redundant recordings at stop signs and intersections. A Gaussian filter of 5 m radius is used to filter out the noisy data. This rough filtering helps reducing the computational workload in some cases by $20\% - 30\%$.

The algorithmic steps will be explained by the help of a small road graph segment with five GPS samples in its vicinity as shown in Figure 3. There is a number of potential prospective snap locations, e.g., at sample station 4, the GPS point could be projected to segment locations $7, 8, 9$. These probable paths moving from one (n) time-stamp to the next $(n + 1)$ can be conceptualized easily via generating a network transition diagram as depicted on the right side of Figure 3. The flow of transitioning from a possible snap location of a GPS sample point to the next sample's possible projection locations can easily be followed using this diagram. The constraints are shown as red crosses; e.g., the path from segment 8 to segment 10 going from station 4 to station 5 should not be allowed since there are no graph connections possible between these prospective locations.

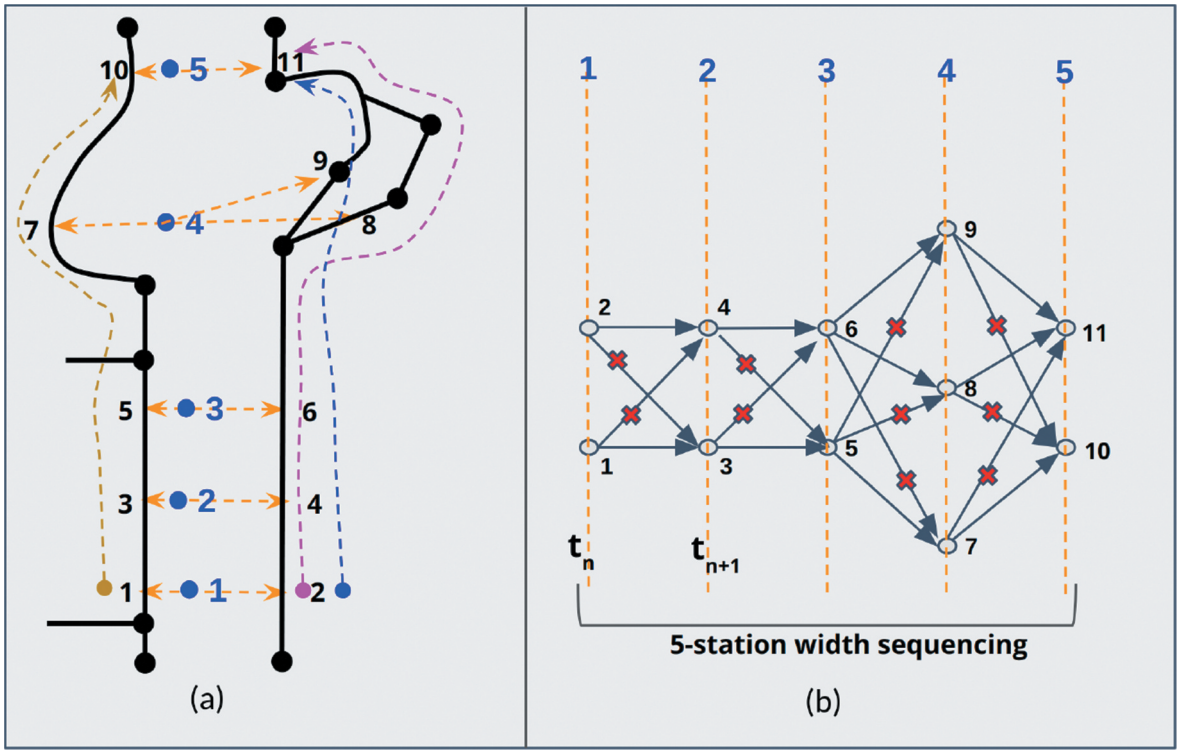

Figure 3. (a) Physical diagram for the road map and GPS samples; blue dots from $1 - 5$ are GPS sample points. Black lines are the actual road network graph segments and the possible snap locations for each GPS point are also depicted as black from $1 - 10$; e.g., GPS sample 4 has three possible nearest segment projections, depicted as $7, 8, 9$. (b) Conceptual network transition diagram across GPS sample stations shown in vertical lines. At each GPS sample location, there is a number of potential prospective snap locations; e.g., at sample 4, the GPS point could be projected to segment locations $7, 8, 9$. The constraints are shown as red crosses; e.g., the path from 8 to 10 should not be allowed since there is no graph connections possible between the two.

There is a number of combinatorial sequences emerging based on all the possibilities of snap locations at each GPS station. The task of finding the probability value of a specific pair (probability of a state) depends on the transitioning probabilities within the pairs in the sequence. The number of digits in a sequence equals to the width of the GPS sample stations in HMC; e.g., there are five digits of the sequence of the case depicted in Figure 3. Mathematically speaking, these probabilities are lumped as a sequence and each sequence has a cost value based on the sum of the shortest path runs aggregated over each pair (See Figure 4). However, shortest path favors the minimum accumulated weight of the edges in the path, and not necessarily those that the GPS points are projected. Hence, the weights are modified proportionally for the nearest segments to the GPS points so that the Dijkstra algorithm implicitly embeds the snapping possibilities and solves for the minimum aggregated sum of the shortest path costs between each pair of the sequence. The modified weights are then reset to their original values when the whole width is shifted by one station to the next batch of width range. The entire algorithm is depicted in the pseudo-code form in Figure 5. This main algorithm is divided into four sections, namely, adapting the width, solving the HMC kernel, sequencing and applying filters and finally detecting the errors for readapting the width before sliding the kernel by one station to the next range as shown in Figure 6.

2.1. Solving mini-dijkstras

A Dijkstra shortest path solver is implemented to run between each pairs of a sequence (Dijkstra 1959). The efficiency in the minimal way of storage and processing speed is very crucial in the overall performance of the MM algorithm, as these mini solves would be running thousands of times during the course of the algorithm execution. The start and end locations could be snapping over the same graph edge in which case, the dijkstra cost optimizer would reduce to the arithmetic operations of finding the proportional weights based on the projection locations along the edge. In fact, there are quite a number of possibilities of how the cost is calculated based on the projection locations of the pair's start and end locations, as shown in Figure 7. The weight w is adjusted as (w^*) based on the snap location l away from v_0 and s is also chosen based on the snap ratio R as the start graph node for the cost optimization solver as computed by Equation (2). The weights on the prospective snap segments are modified to make sure that the cost optimization solver would have a proportional and ensured bias on the close segments to the GPS points. MM routing cases are found to be not particularly sensitive to the weight modification heuristics. For the cases tested within

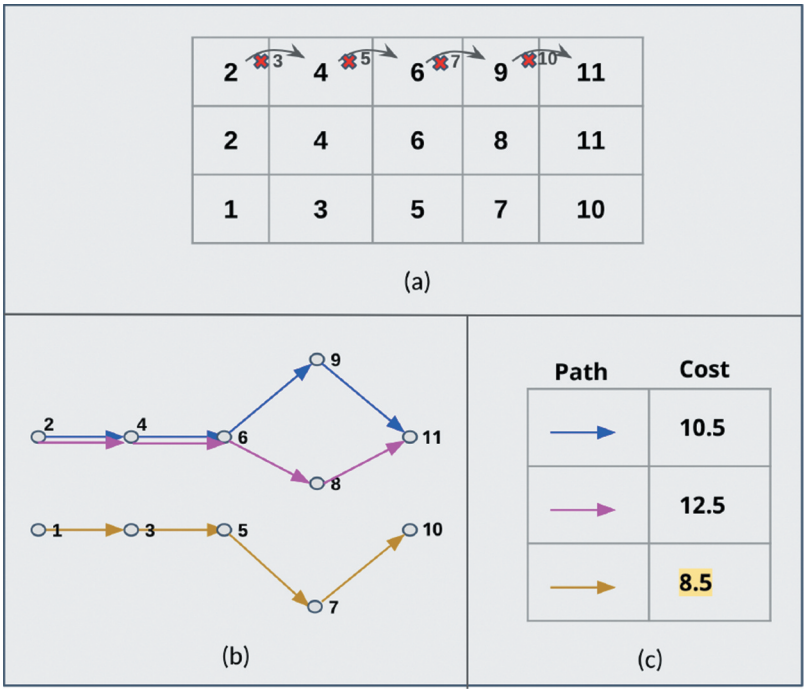

Figure 4. Finding the minimum cost sequence; (a) the possible sequences in five-digit combinatorials of snap locations numbered consecutively across GPS samples. The sequence generation algorithm prunes the possibilities based on the topological connections; number 2 at the first station transitioning to the number 3 at the second station is not allowed and filtered from the sequence as depicted with the crosses. (b) three probable path sequences. (c) the cumulative cost of mini-dijkstra runs aggregated between the pairs of each sequences; minimum cost sequence is 1 — 3 — 5 — 7 — 10 with the cost of 8.5.

Algorithm 1 Main Algorithm

```
1: procedure SOLVE(gps, trip, G(V, E), weights, width)
2:     adapted ← false
3:     original ← width
4:     for i ← 0 to gps.size() do
5:         if adapted then
6:             width ← adjustWidth(width)
7:             adapted ← false
8:         else
9:             width ← resetWidth(original)
10:        end if
11:        n ← 0                                              Adapt Width
12:        filtered ← 0
13:        while n < width do
14:            m ← n + 1
15:            geoDist ← geodesic(gps, i, n, m)
16:            weights ← adjustWeights(gps, i, n) {only for edges closest to point ni}
17:            weights ← adjustWeights(gps, i, m)
18:            for j ← 0 to closestEdges(gps, i, n) do
19:                start ← snap(closestEdge(gps, i, n, j))
20:                for k ← 0 to closestEdges(gps, i, m) do
21:                    end ← snap(closestEdge(gps, i, m, k))
22:                    cost ← costOptimizer(G, start, end, weights)
23:                    if cost > maxCost then
24:                        filtered ← pair(j, k)
25:                    else
26:                        pathSegments ← register(start, end, cost, geoDist)
27:                    end if
28:                end for{k}
29:            end for{j}
30:            weights ← resetWeights()
31:            n ← m                                          Solve Kernel
32:        end while
33:        sequences ← generateSequences(i, width, filtered)
34:        minCost ← maxFloatCost
35:        for sequence in sequences do
36:            sequenceCost ← 0
37:            for path in sequence.pairs() do
38:                pathSegment ← pathSegments(path.first, path.second)
39:                sequenceCost ← sequenceCost + pathSegment.cost
40:            end for
41:            if sequenceCost < minCost then
42:                minSequence ← sequence
43:                minCost ← sequenceCost
44:            end if                                         Sequencing
45:        end for{sequence}
46:        if minSequence exists then
47:            path ← minSequence.path()
48:            pathDistance ← path.computeDistance(path[0], path[1])
49:            if (!adapted and pathDistance/path.geoDistance > maxDesicRatio)
50:                adapted ← true
51:                i ← i − 1
52:            else
53:                trip ← trip + path                         Slide Kernel Feedback
54:            end if
55:        end if
56:    end for{i of gps}
57: end procedure
```

Figure 5. Main Hidden Markov chain algorithm.

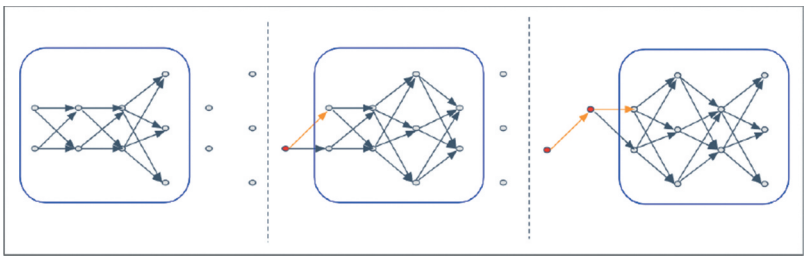

Figure 6. Sliding the kernel by one station.

our work, we have found out that one tenth of the original weights is adequate in forcing the solves to follow the GPS samples. Another crucial observation is that the Markov characteristics of the algorithm seem to be relaxing the sensitivity on the ad-hoc nature of the weight factor selection schemes.

$$R = l/L$$

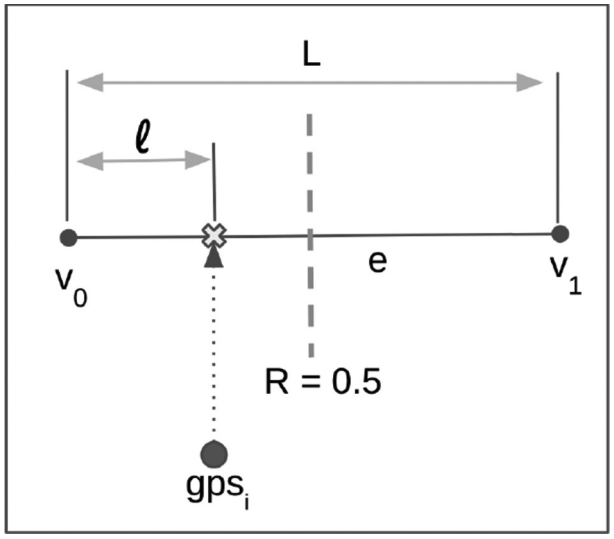

Figure 7. Projection of a GPS point GPS_i on a prospective edge e.

$$s = \begin{cases} v_0, & R \leq 0.5 \\ v_1, & \text{otherwise} \end{cases}$$

$$w^* = \begin{cases} R \cdot w, & R \leq 0.5 \\ (1-R) \cdot w, & \text{otherwise} \end{cases} \quad (2)$$

In general, the Dijkstra Condition (DC) on each vertex v_i can be specified by Equation (3) which states that the cost d_i can not be greater than the minimum of the cost of any incoming vertices connected to v_i via the edge's weight w_{ij}. The DC condition is satisfied in a breadth first search manner by the Dijkstra-$\mathcal{D}$ kernel originated from the source (start) node and terminated at the destination (end) node. We opt for using a priority queue implementation for the Dijkstra solver (Felner 2011) which seems to supersede parallel implementations (Wang et al. 2017) in speed due to its small sub-graph size between the start and the end nodes; in MM case, pairwise GPS timestamps are at most 200 m away, i.e., only a few hundred edges needed for traversals at the most.

$$d_i = (d_j + w_{ij}) | w_{ij} : v_j \mapsto v_i, \in N(v_i)$$

$$D_{start,end} = \min_{v_i \in G(V,E)|_{start}^{end}} (d_i) \quad (3)$$

2.2. Sequencing

The essential juxtapose of the algorithm pivots around the ability of generating the sequences under the topological constraints of the road network. The latter is applied as filters in the sequence generation engine. The filtered pairs are identified by the failure of the cost optimization solver. The sequences are generated after the exhaustive solver cycle so that the filtered pairs could be applied simply as constraints on the combinatorial number sequence generation scheme (See Figure 4). The cost between each pair of indices found by the solver in each sequence is aggregated and the total cost is paired with the sequence number as depicted in Equation (4). The optimization problem is then reduced to picking the minimum total cost sequence among all probable sequences. When the minimum cost sequence is picked as shown in the example as the second sequence with the cost of $C_2 = 8.50$, only the first point in the sequence is set for the sure-match and the kernel is shifted to the right (next time-stamp station), thereby the decision for the current sample's snap location (state) is always made based on the probability states of the GPS sample stations in the next range whose width is not constant. So, the sequence s_2 is picked and the first GPS sample location in the sequence is fixed at the snap location corresponding to the index $\{\mathbf{1}\}$ as depicted in Equations (4) and (5).

$$cost_k : C_k = \left\{ \sum_{i=0}^{width-1} \left\| D_{i,i+1} \right\| \right\}_k$$

$$s_0 = \{2,4,6,9,11\} : C_0 = 10.5$$

$$s_1 = \{2,4,6,8,11\} : C_1 = 12.5$$

$$\boldsymbol{s_2}{=}\{\mathbf{1},\mathbf{3},\mathbf{5},\mathbf{7},\mathbf{10}\} : C_2{=}\ \mathbf{8.50} \quad (4)$$

$$argmin(s_k, C_k) \mapsto \{s_2, C_2\} \quad (5)$$

2.3. Adapting width

The decision of adapting the MC window width is based on the ratio between the geodesic distance of the route (snaps) to the actual geodesic distance of the GPS points (samples) similar to the probability density function of Newson and Krumm (2009). However, instead of the difference depicted in Equation (1c), we have used the ratio of the distances to detect the errors in the map-matching process. The need for increasing the kernel width can easily be noticed in the mid section of a typical example case shown in Figure 8 as the ratio of the geodesic distances exceeds an ad-hoc threshold limit of ten ($\times 10$). Basically, the kernel's width was not wide enough to include the future history that would anticipate favoring on a more logical (sensical) map-matching. Hence the redundant looping around the intersection is avoided with the help of including more points inside the MC kernel as seen in Figure 9 (10 points versus 5 points). The adaptation scheme of doubling the width has a maximal ceiling of 14 points and will not re-try once this ceiling is reached as the number of probable sequences quickly becomes formidable to enumerate and solve.

The adaptive selection of the width in the MC kernel has shown to have hardened the results tremendously with tolerable computational cost increase

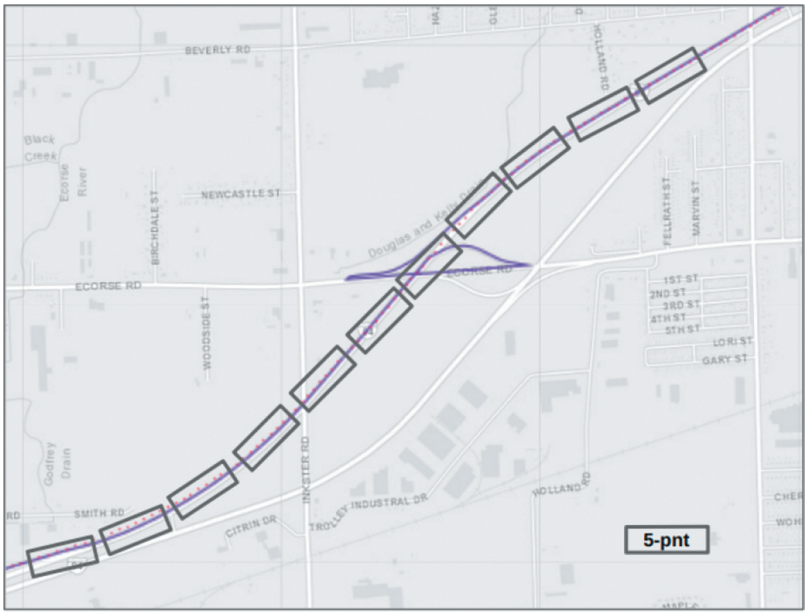

Figure 8. Map-matching result using the fixed width MC kernel of five GPS points across; cost optimization mistakenly chooses the route with a redundant loop in the middle of the trip routing. Red dots are the actual GPS points.

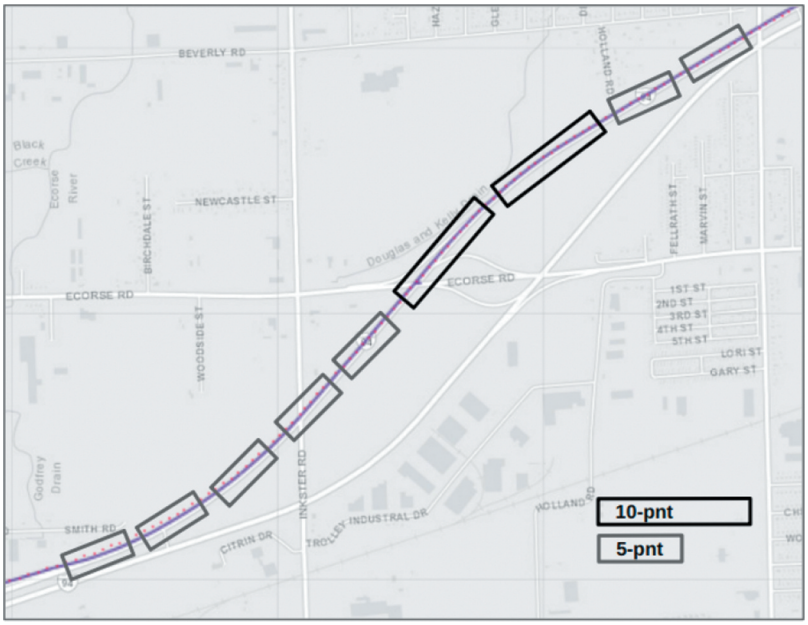

Figure 9. Map-matching result using the adaptive width MC kernels that changes; the error is detected since the ratio between the geodesic distance of current route to the actual geodesic distance between the pair of GPS samples is more than the defect threshold. The MM is fixed adaptively using a wider width of 10 points.

since the feedback loop depicted in the main algorithm illustrated in Figure 5 reverts back onto the original and narrower width of MM right after running the wider kernel scenario. From the computational experiments, it could also be speculated that the likelihood of making an erroneous decision with a narrower span is not uncommon particularly for the cases where the sampling frequency is not adequate and/or more valid sequences exist.

Another example for the adaptive kernel solving the wrong path selection issue due to the erroneous GPS samples (latitude shift) can also be seen in Figure 10, where the result of our adaptive scheme is compared against the fixed width algorithms (Newson and Krumm 2009; Brakatsoulas et al. 2005; Felner 2011; Greenfeld 2002). Adaptively switching the width twice more than the nominal value helped including the "key" GPS samples whose projection snaps are over the correct graph road segment. The automatic switch from fixed width to adaptive width is detected by computing the ratio between the geodesic and the route distances within any consecutive pairs of GPS samples during the "Slide Kernel Feedback" step of the main algorithm depicted in Figure 5.

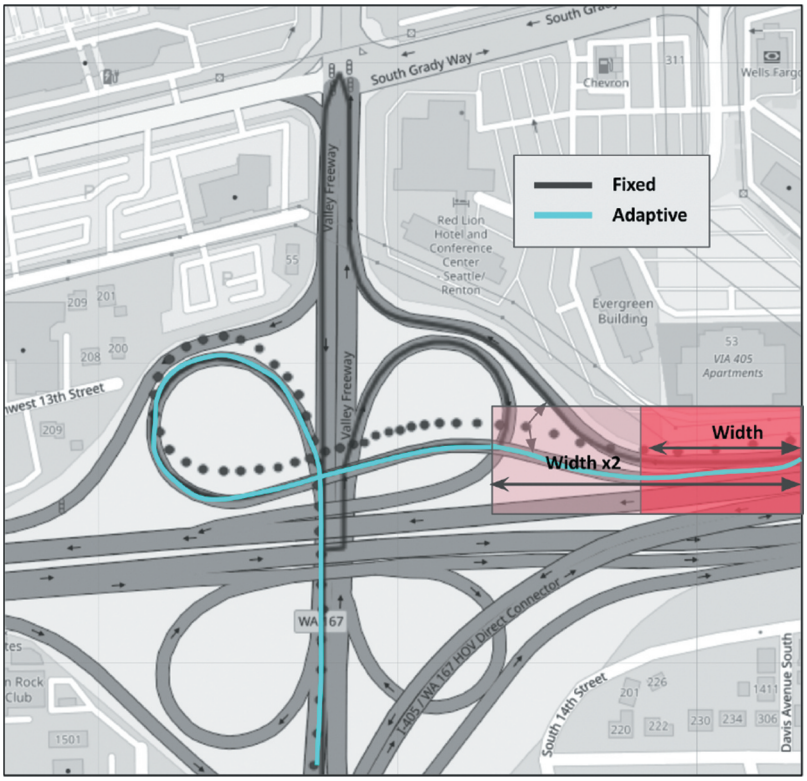

Figure 10. The comparison of our adaptive kernel width algorithm vs fixed width conventional map-matching algorithms. Black dots denote the input GPS samples that are erroneous due to a random latitude shift. The fixed width algorithms result in dark gray path, with redundant loops indicative of "faulty" map match. The adaptive switch of the kernel width twice more than the fixed width enabled HMC algorithm to include the "key" GPS samples to change the overall route to a more correct alternative as depicted by the cyan path.

3. Results and discussion

We have used thousands of sets of GPS trip data, emitted from the test vehicles across the continental US to verify and harden the results of our algorithm. In fact, the remedying idea of adapting the Markov chain width directly came from analyzing these valuable sets of data and seeing where the problem areas arise. The observed cases where we have noticed a potential adaptation needed can be itemized as follows:

- Approaching to the forking road segments where GPS samples are actually closer to the wrong section of the fork is resolved to a "correct" path only by "looking ahead" toward the next set of points (See Figure 11, Figure 12, Figure 14).
- The noise in the GPS data is making many choices viable and only having a cumulative score of combinatorial path optimization leads to a reasonably "correct" path. (See Figure 15).
- The upper and lower passes of the road network where the z-level changes may have overlapping two dimensional (Lon,Lat) coordinates and hence graph edges built without hashing on z-level values, can be connected from lower to upper sections. The resolution may both need to have the graph to be z-aware and also use wider markov chains in map-matching. (See Figure 10, Figure 11, Figure 14).
- The round-about sections require a wider kernel widths to identity the paths correctly. (See Figure 13).

All of the above observations show the need for a Markov Chain type optimization to be employed and also reflects the need to change the kernel width adaptively to resolve the paths correctly.

The sequences generated via Markov chains uses many mini-Dijkstra runs as explained in the Subsection 2.2 above. However, if the GPS points are too far away from each other, particularly the case for the inability to emit the data inside tunnels, etc., the Dijkstra runs should be allowed to cover more than the distance between the two GPS end points. In order to minimize the computational load, however, if the Dijkstra runs can not reach the "end" node from the "start" within a user set limit, e.g., within 10 hops, then the sequence is deemed to be not valid, and skipped by leaving gaps in the matched route (See Figure 11).

There are also rare scenarios due to numerical instability induced by the modifications of the edge weights proportional to the proximity of the graph edges to the GPS samples, that can result in sequences that might have redundant back and forth traveling over the same edge. These are referred as folding vertex paths and should be

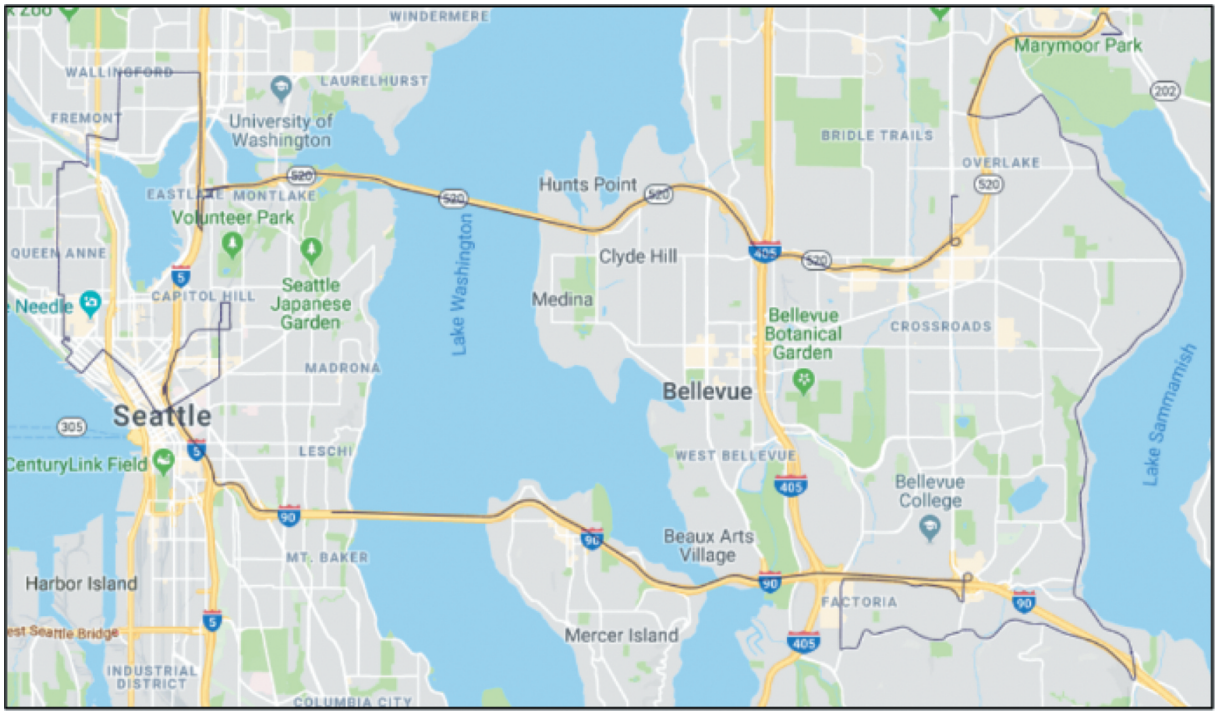

Figure 11. Result of map-matching over the Seattle data set courtesy of Microsoft (Newson and Krumm 2009).

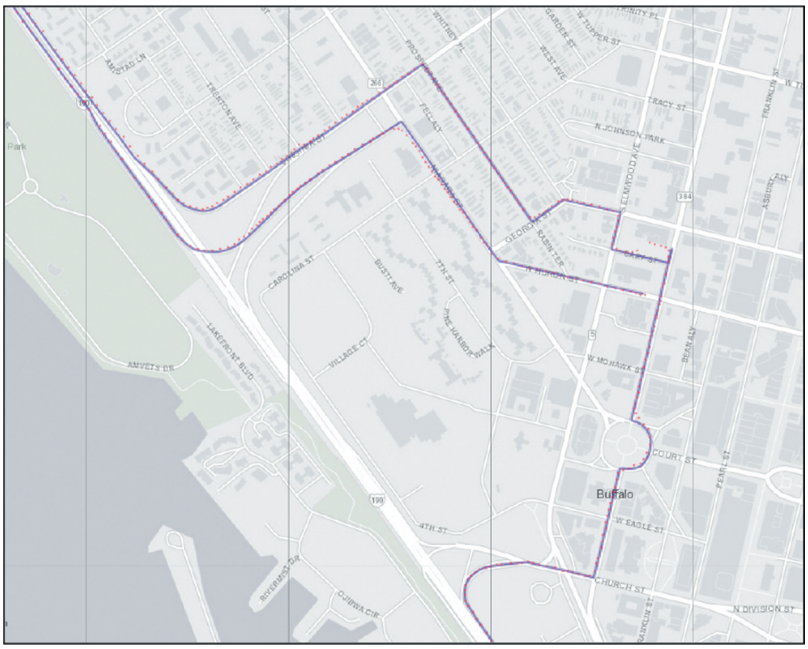

Figure 12. Result of map-matching; the trip data is provided by Ford motor company. Red dots depict the GPS sample points emitted from the vehicle – continuous blue line is the result of the map-matched route.

filtered out before finding the minimum cost sequence depicted by Equation (4). Filtering out these paths requires additional computational time, and even though the individual check is not expensive, the need to repeat the check for shifting kernels can not be amortized easily. Hence, filtering is made optional, as a typical trade-off between accuracy and speed. The additional time is proportional to the kernel width, the total number of GPS points, as well as the number of prospective snap edges per GPS point found by the R-Tree search. It can vary from low $1\%-2\,\%$ to almost as high as $30\,\%$ for the rare cases of $10+$ hour-long trip durations with $2-5$ seconds GPS frequencies. For example, the folding vertex paths, such as the sequence as, $\{2,2,2,2,2,2,3,3,4,4,4,3,5,5,6,7,7\}$, can easily be identified due to its redundant $\{\ldots 3,4,4,4,3\ldots\}$ fold pattern.

An ad-hoc matching score per trip is computed to understand how well the mapped route is matching with the underlying GPS points as depicted in Figure 16. The GPS points are snapped back onto the map-matched

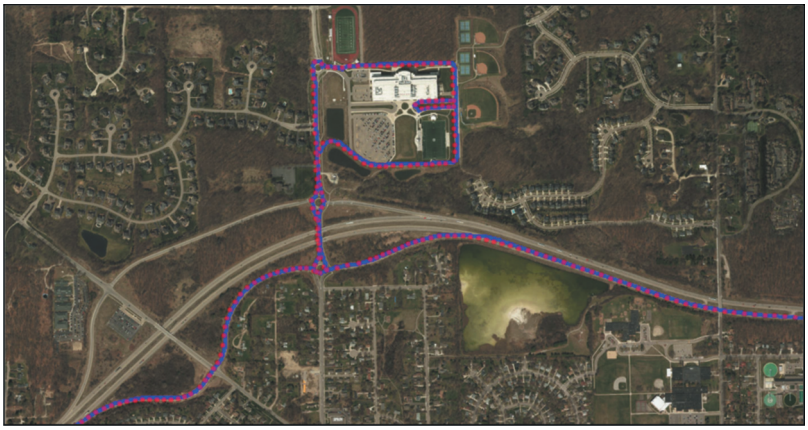

Figure 13. Result of map-matching; the trip data is provided by Ford motor company. The matched route shown in blue most likely depicts a person's dropping kids off at the school grounds and returning back making turns around the multiple roundabouts.

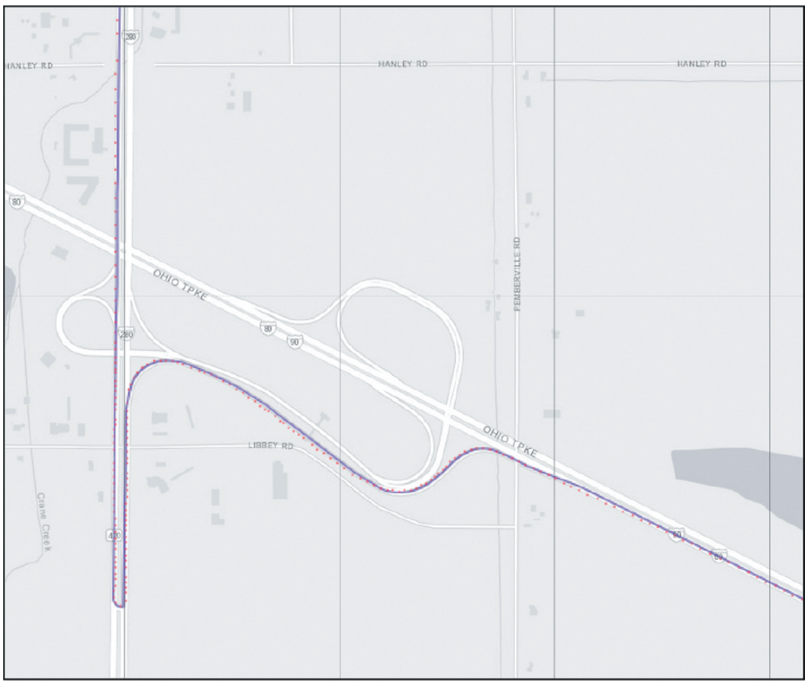

Figure 14. Result of map-matching; the trip data is provided by Ford motor company. Map-matching correctly determines the "correct" path at forking intersections with blue line as matched route and red dots as raw GPS points.

route and the distance difference between the GPS location to its closest snap location is aggregated over the whole set and divided by the total number of GPS sample points. In lieu of the ground truth data which is unfortunately not available for our tests, this heuristic measure at least gives a notion for an error of the match similar to mean squared residuals over the entire data set. Almost all good matches demonstrate a match score well within $1-5$ m (1 m $\approx 10^{-5}$ in degrees). Any match score higher than this threshold range indicates that either the samples are highly noisy and/or the order in the GPS timestamps are erroneous.

Finally, the performance of our algorithm can be said to be dependent on a number of parameters; total number of GPS points, the input parameter of initial Markov Chain width (the adaptive kernel never goes below this preset value), the number of closest edges per GPS point found by the R-Tree. This latter parameter and the chain width directly impacts the total number of combinatorial sequences and consequently the number of mini-dijkstra runs. Our algorithm allows users to trade accuracy versus speed. However, it is rare that we have seen a map-matching process taking more than a few seconds on modern laptops (2–3 GHz intel i7 processors). Another limiting parameter we have instrumented is the cap on the number of combinatorial sequences generated, currently, at $10,000$ that gets generated within each kernel shift that is usually never violated with the common widths of $6-9$ points and $3-4$ closest number of edges per GPS sample.

The testing results shown in Figure 16 are tabulated on Table 1, including $140K$ GPS samples of 77 trips

Figure 15. Noise in the GPS samples is quite visible by the void red circles zoomed in on a portion of the Microsoft's Seattle data set. The Map-matching algorithm finds the best route by screening possible path sequences under the constraints of the graph road network topology.

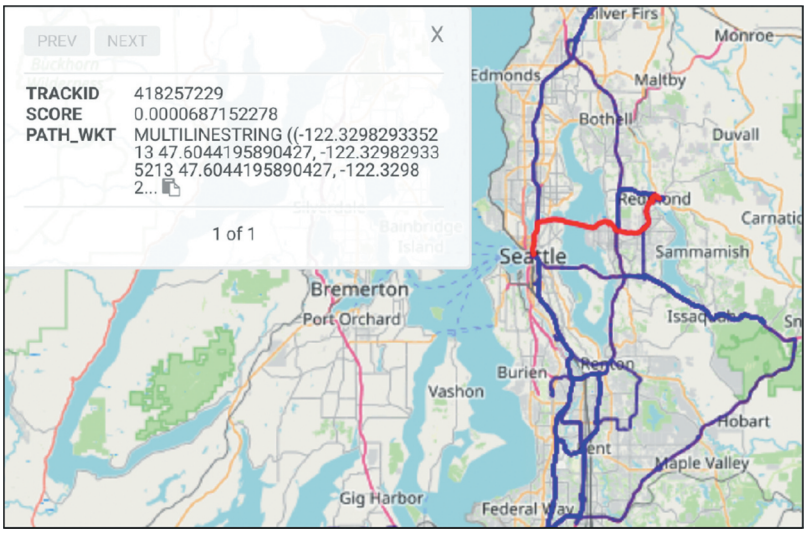

Figure 16. Result of map-matching; the trip data is provided by Ford motor company. The picture shows the map-matching routes for 77 trips around seattle area. The red line specifically depicts a user picked trip with an id of 418257229 and a match score of $\approx 1.2 * 10^{-5}$.

Table 1. Adaptive versus fixed width results.

Width (base)	Method	Time/s	AvgErr/m	MaxErr/m
	Fixed	5.23	11.6	152.7
5	Adapt	9.54	1.9	5.5
	Fixed	17	6.8	152.7
8	Adapt	25	1.1	1.3
10	Fixed	53.7	2.9	152.7
14	Fixed	453.6	0.9	1.3

with varying frequencies computed over a graph of $475K$ edges, and a maximum of 3 closest snaps per GPS sample within a search radius of 10 m. The table lists mean and maximal errors found in these trips for various base widths to compare the fixed width algorithms against our adaptive method. Mean squared residuals over all trips are tabulated over the "AvgErr" column and the maximum error found in any trip depicted as the "MaxErr" column, respectively. One major noteworthy remark on these results is that there appears to be one trip case that has a wrong map match causing over hundred meter error and none of the fixed widths up-to 14 points could address the trip's map successfully (within an admissible error bound). However, the fixed HMC width of 14 takes almost exponentially large run time increase compared to the lower widths, that makes the entire map-matching process computationally prohibitive. The solution is the use of our adaptive algorithm and as depicted over Table 1, even the use of only 5 (five) points adaptively was seemingly able to switch to the necessary width where necessary to resolve the erroneous path matching issue. The optimal choice seems to be the use of base width of

8 adaptively to suppress both the maximal and average errors. Both 8 and 5 point adaptive results show orders of magnitude speed gains compared to the fixed width conventional HMC algorithms.

Though the entire algorithm depicted in Figure 5 is not particularly suitable for parallelization, we have parallelized the batch runs of many trips as well as registering the mini-dijkstra runs within each solver. On one test batch consisted of more than $300K$ sample points belonging to 370 individual trips of varying degrees of sampling frequencies between 0.5 seconds and 5 seconds, we were able to obtain results in less than 24 seconds using 8 cores where 95 percent of the trips had match scores well below 1 m over a graph of approximately 7 million edges. Though these results are satisfactory in practice and the algorithm has been adopted by our industry partner car company, parallel batch runs can not be proven to provide linear at-scale performances. One of the reasons of bottlenecking the overall scalability is due to the "heavier" sets of trips with more GPS samples that might also require more adaptive cycles. In the future, we are planning to improve the scalability of the algorithm by addressing parallelism within each individual module in the core algorithm depicted in Figure 5. Finally, the novel idea of using the adaptive Markov kernel width proposed as the main contribution of our paper could easily be adopted and plugged into the existing map-matching algorithms to improve the general accuracy of the results.

It is also worth mentioning about the nature of the software in this algorithmic work, that is entirely implemented from scratch using C++, and without the use of any third-party libraries. The I/O to the map-matching algorithm is provided from the in-memory database of Kinetica, a GPU streaming data warehouse and using its propriety C++ APIs. The map-matching is itself a yet another Kinetica API, and available in RESTFUL/C++/Java/R/Python API formats. The novel idea of using an adaptive Markov chain width in addressing map-matching problems is also granted a provisional US Patent recently (EFSID: 38,512,015, App No: 62,970,845).

Disclosure statement

No potential conflict of interest was reported by the author(s).

Notes on contributors

Bilge Kaan Karamete is the Senior Director of Engineering for the Geospatial, Graph and Visualization efforts at Kinetica. His research interests include computational algorithm development, unstructured mesh generation, parallel graph solvers and computational geometry. He holds PhD in Engineering Sciences from the Middle East Technical University, Ankara Turkey, and post doctorate in Computational Sciences from Rensselaer Polytechnic Institute, Troy New York.

Louai Adhami is a principal engineer at Kinetica, and holds a PhD in robotics from INRIA. He works on high concurrency graph solvers and graphics capabilities. He enjoys doing software architecture for distributed systems and teaching at George Washington University, Washington DC.

Eli Glaser is VP of Engineering at Kinetica. He leads the development teams concentrating in data analytics, query capability and performance. Eli holds Master's in Electrical Engineering from The Johns Hopkins University, Baltimore Maryland.

Data availability statement

Due to the nature of this research, participants of this study did not agree for their data to be shared publicly, so supporting data is unfortunately not available.

References

Boeing, G. 2017. "OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks." *Computers, Environment and Urban Systems* 65: 126–139. doi:10.1016/j.compenvurbsys.2017.05.004.

Brakatsoulas, S., P. Dieter, S. Randall, and W. Carola. 2005. "On Map-matching Vehicle Tracking Data." *Pages 853–864 of: Proceedings of the 31st International Conference on Very Large Data Bases*. Trondheim, Norway.

Chen, B. Y., H. Yuan, Q. Li, W. H. K. Lam, S.-L. Shaw, and K. Yan. 2014. "Map-Matching Algorithm for Large-Scale Low-Frequency Floating Car Data." *International Journal of Geographical Information Science* 28 (1): 2238. doi:10.1080/13658816.2013.816427.

Dijkstra, E. W. 1959. "A Note on Two Problems in Connexion with Graphs." *Numerische Mathematik* 1 (1): 269271. doi:10.1007/BF01386390.

Felner, A. 2011. Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case against Dijkstra's Algorithm. *In: Proceedings of the Fourth Annual Symposium on Combinatorial Search, Castell de Cardona, Barcelona, Spain.*

Goh, C. Y., J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, and P. Jaillet 2012. "Online Map-matching Based on Hidden Markov Model for Real-time Traffic Sensing Applications." *Pages 776–781 of: 15th International IEEE Conference on Intelligent Transportation Systems*. Anchorage, AK, USA.

Greenfeld, J. 2002. "Matching GPS Observations to Locations on a Digital Map." In *Proceedings of 81th Annual Meeting of the Transportation Research Board*. Washington, DC, USA.

Guttman, A. 1984. "R-Trees: A Dynamic Index Structure for Spatial Searching." *Page 4757 of: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*. Boston, Massachusetts, USA: SIGMOD 84.

Karamete, B. K., R. Aubry, E. L. Mestreau, and S. Dey. 2016. "A Novel Double Link Structure (DLS) with Applications to Computational Engineering and Design." *AIAA Aerospace Sciences Meeting* 54: 1301.

Kinetica. 2020. *Kinetica DB. Inc Document for Network Graph Solvers - V7.0.11*. https://www.kinetica.com/docs/graph_solver/index.html. Accessed 06 01 2020.

Newson, P., and J. Krumm 2009. "Hidden Markov Map Matching through Noise and Sparseness." *GIS '09: 17th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 336343. Seattle, Washington, USA.

Quddus, M. A., W. Y. Ochieng, and R. B. Noland. 2007. "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. doi:10.1016/j.trc.2007.05.002.

US-Government. 2020. "US Government GPS Accuracy." Accessed 06 January 2020. https://www.gps.gov/systems/gps/performance/accuracy/

Wang, Y., Y. Pan, A. Davidson, Y. Wu, C. Yang, L. Wang, M. Osama et al. 2017. "Gunrock: GPU Graph Analytics." *ACM Transactions on Parallel Computing* 4 (1): 1–49. doi:10.1145/3108140.

Wei, H., Y. Wang, G. Forman, Y. Zhu, and H. Guan. 2012. "Fast Viterbi Map Matching with Tunable Weight Functions." *Page 613616 of: Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. SIGSPATIAL 12. New York, NY, USA: Association for Computing Machinery.

White, C. E., D. Bernstein, and A. L. Kornhauser. 2000. "Some Map Matching Algorithms for Personal Navigation Assistants." *Transportation Research Part C: Emerging Technologies* 8 (1–6): 91–108. doi:10.1016/S0968-090X(00)00026-7.

Xin, R. S., J. E. Gonzalez, M. J. Franklin, and I. Stoica 2013. "GraphX: A Resilient Distributed Graph System on Spark." In *First International Workshop on Graph Data Management Experiences and Systems*. GRADES 13. New York, NY, USA: Association for Computing Machinery.