AI Assisted Trail Map Generation based on Public GPS Data

Jared Macshane, Ali Ahmadinia

Department of Computer Science and Information Systems
California State University San Marcos, CA, USA

Email: jmacshane@csusm.edu, aahmadinia@csusm.edu

***Abstract*—Hiking trail maps are typically created manually by survey, a time-consuming process. This process is expensive and must be repeated over time to improve accuracy. This paper proposed an inexpensive, automatic, and accurate trail network generation method from anonymous public GPS data utilizing a growing self-organizing map (GSOM). This technique does not rely on sequential GPS traces to learn network topology, unlike other approaches. Tuning several hyper-parameters can adjust this process for datasets and networks with unique characteristics. Reconstruction and adaption are also possible based on newly acquired data sources. Constructed trail maps, trained on publicly available GPS data, are compared against a ground truth map from Open Street Map (OSM). Performance is evaluated based on completeness, accuracy, and topological correctness. Testing on sparse networks with minimal GPS noise suggests favorable performance.**

Keywords - growing self-organizing maps; GPS data; trail network construction.

I. Introduction

Trail networks are topological maps that are needed for navigation [1]. Trails consist of path segments that may cross at intersections points. Successful navigation of these paths is critical for the enjoyment and safety of recreational and SAR (search and rescue) hikers.

Historically, networks for trails have been created manually by surveyors and cartographers with specialized equipment. While typically accurate, these methods remain expensive and time-consuming [1].

With the advent of inexpensive GPS recordings devices such as smartphones and smartwatches, there has been a strong interest in the development of trail network maps utilizing this data.

OpenStreetMap (OSM) has become a popular platform for collaborative crowd-sourced mapping. Users can edit streets, sidewalks, trail paths, and any feature relevant to cartography. Users may also upload public GPS traces which could lead to irregular and noisy data. Other problems with these OSM public trace datasets include limited geographic representation, cheaper devices, and people wandering off the trail.

In this paper, we construct trail networks using a growing self-organizing map and apply connection and contraction rules. Previous work using GSOMs to approximate principal curves and work using SOMs to construct road networks motivates this methodology.[2], [3]

Hashemi [4] establishes benchmark metrics to evaluate various network construction methods. These include completeness, precision, and topological correctness metrics. They construct networks using Kernel Density Estimation from OSM traces and report metric evaluation. A portion of this paper will compare our method's performance on these metrics on the same dataset.

II. Background

A. Method Archetypes

Road and trail inference can be done from two data sources, aerial imagery or GPS data recordings from mobile devices. While algorithms utilizing imagery can be productive, they can suffer from a lack of consistent high-quality data sources due to costs. Additionally, physical features can introduce errors into the system through visual obstruction and variant weather conditions. These methods are covered by authors [5], [6]

Alternatively, other algorithms infer road networks from GPS data recording, often from inexpensive mobile devices such as a smartphone or watch. This allows for a plethora of data; however, GPS errors can lead to errors in network inference. This is the data inference source primarily covered in this paper.

Four popular method archetypes for road and pedestrian path network construction from GPS data sources exist: Point clustering, Kernel Density Estimation, Trace merging, and Intersection Linking, each with unique advantages and drawbacks [7].

1) Point Clustering: Fundamentally, a clustering algorithm will construct a representative set of points from the input data that best describe the underlying geometry. K-means have been used to find the positions of these representative nodes. The approach used in this paper is a GSOM, similar to K-means. Typically Euclidean or Vincenty distances (for longitude and latitude coordinates)

are used as a distance metric. Once a representative set has been formed, there is a linking step to form the network's topology [8]. Methods vary on how to form edges between nodes. One method utilizes trajectory (non-anonymized) based GPS traces where information from GPS trajectories is used to form links between representative nodes. For example, Stanojevic et al. [9] use the angle of the heading embedded inside the GPS trajectories to form edges.

The method presented in this paper is an example of a point clustering algorithm, implementing a new linking component since we utilize anonymized trajectories.

Yun [3] uses a growing self-organizing map to construct a road network from satellite imagery. Nodes are formed by neurons adapting to the different colors of a road from its background and edges formed by pattern matching. The linking step is matching structure to predefined templates.

2) Kernel Density Estimation (KDE): Typically in KDE algorithms, a probability density function of trajectories is created first. By creating an image with a pixel value corresponding to the trajectory intersection, a discrete image of the data is produced. Then a threshold is applied to remove errors and outliers. Road line outlines are then computed in a variety of ways, among them being Voronoi diagrams. From road outlines, intersections are extracted, yielding a final road network [8].

Yang [10] used the common Kernel Density Estimation method-based point clustering algorithm to generate pedestrian paths based on filtered traces. A classification schema is used to filter traces into categories that serve different purposes in network construction.

3) Trace Merging: A markedly different family of algorithms, **Trace merging** algorithms aggregate traces into a continuously growing network. New trajectories are iteratively presented to the network. If this trajectory has no overlap with the existing network, it can be added entirely. For overlapping sections, merging is handled by changing the weights of edges. Network geometry and topological assumptions are made. As each trace is added, new network construction must satisfy existing network constraints. This method allows for active learning [8].

As these algorithms explicitly require trajectories, they cannot be applied to anonymized trajectories.

4) Intersection Linking: This approach focuses on detecting intersections first and then linking them together using trajectory information[7].

Karagiorgou [11] used changes in the heading to identify their intersection points.

The outline of this process is as follows: Extract points from input trajectories that satisfy specific heading (turning) or speed changes (slowing down). Cluster these points using k-means or other methods, create one node for each cluster, and label it as an intersection. Create links between intersections from trajectory data that hits multiple intersections.

5) GSOM Method: Our GSOM method follows a similar structure to many point clustering algorithms with a representative point finding algorithm and a subsequent linking step. The following presented algorithm distinguishes from many common methods by utilizing anonymized traces, thereby allowing for more public datasets, such as from OSM.

III. Methodology

The core of this method is the utilization of a growing self-organizing map (GSOM). A GSOM is a variant of the self-organizing map, which is a form of a neural network. Given a SOM and a dataset, a data point is presented to the network and one neuron is chosen as the winner. A weight adaption is shared by the winning node and other nodes within some distance neighborhood.

Rather than a fixed number of neurons, the GSOM can grow new neurons when presented with a data point that is not sufficiently represented.

This method consists of 5 major phases: initialization, growth/learning, applying connection rule, network contraction, and finally smoothing.

A. Initialization

The GSOM is initialized with 3 nodes, whose weights are set to random points from the dataset.

The hyperparameters spread factor SF and neighborhood radius R are set. From the spread factor, a growth threshold GT is calculated. s

$$GT = -2 * \ln SF \tag{1}$$

B. Growth & Learning

This portion of the algorithm consists of learning, i.e. weight adaption, and growing new neurons. Datapoints ($p \in D$ dataset) are presented to the network N with neurons $n \in N$ one at a time. Where d represents the Euclidean distance between p and n. p_1, p_2 are the x and y values respectively.

$$d = p - n = \sqrt{(p_1 - n_1)^2 + (p_2 - n_2)^2} \tag{2}$$

Algorithm 1 outlines the learning and growing phase of our method. Note the terminal condition is when no weights have been changed.

1) Find Winner: A fundamental part of all SOMs is competitive learning. In this function, we calculate the error, or distance to the datapoint (2) and retrieve the neuron with the smallest value. This neuron is considered the winning node. For example, we compare the distance from datapoint p, to two neurons n_1, n_2. Whichever neuron is closest becomes the winning node.

In a brute force method, finding a winner takes $O(m)$, where m is the number of neurons, for each data point. Taking n to be the number of data points. If each learning

Algorithm 1 LearnAndGrow

```
1:  procedure LearnAndGrow(N,D,r)                ▷
    N is the GSOM network, D cleaned dataset, and r
    neighborhood radius
2:      ΔW ← ∞
3:      while ΔW! = 0 do
4:          ΔW ← 0
5:          for p ∈ D do
6:              winner,error ← FindWinner(N,p)
7:              ΔW ← AdaptWeights(winner,p)
8:              if error ≥ GT then
9:                  GrowNode(N,p)
10:             end if
11:             N.iteration += 1
12:         end for
13:     end while
14: end procedure
```

epoch runs through the entire dataset and finds a winner then each epoch takes $O(mn)$.

As the map grows, more neurons are added scaling m. In order to speed up this process, we utilize a k-d tree of neurons. We can query the k-d tree with $O(\log m)$ on average [12].

Therefore, for each epoch, this method takes on average $O(n \log m)$, a marked improvement for scaling to larger networks.

2) Adapt Weights: Another key feature of SOMs is weight adaption. Weights are adapted to all nodes within a specified radius of the winning neuron, which is determined by the weight adaption rule below. As seen in Fig. 1, we have a winning neuron n_1, yet neuron n_2 changes weight since it lies within r distance from n_1 and n_3 remains unchanged.

Note k is the iteration value, starting at 0 and incremented after each data point is processed. See line 9 in Algorithm 2.

$$n_j(k+1) = \begin{cases} n_j(k) & n_j \notin M \\ n_j(k) + LR(k,N) \times d(n_j, p) & n_j \in M \end{cases} \tag{3}$$

Where n_j are the weights of neurons in the network, M is the set of neurons in the neighborhood of the winning neuron. LR is a function that determines the current learning rate value. The current learning rate is determined by the following equation

$$LR(k,N) = 0.02^{\lfloor \frac{1+k}{|N|} \rfloor} \tag{4}$$

Where $|N|$ represents the number of neurons in the network. The Learning Rate is a small number, much less than 1, raised to the floor of the fraction of the current iteration to the total number of neurons.

3) Grow Node: If the error is greater than the growth threshold than that data point is not sufficiently covered within a certain radius of an existing neuron. We grow a new neuron whose weights are assigned to the data point.

$$n_{\text{new}} = p_i \tag{5}$$

While n_{new} may be placed initially in an inaccurate position from a noisy p_i, with subsequent learning iterations and smoothing, this point should achieve a more accurate final weight since from the initial data cleaning we know there are additional data points within a certain distance.

C. Connection Rule

After node growth has concluded and the terminal condition is met, the connection rule is applied. Again, each data point p_i is presented to trigger the connection rule. However, each time we consider the two closest neurons to the input data point and form an edge between them. This is inspired by the principle of Hebbian learning "neurons that fire together wire together" [13].

As seen in Fig. 2, neurons n_1, n_2 are closer to data point p than n_3 and thus we add an edge between n_1 and n_2.

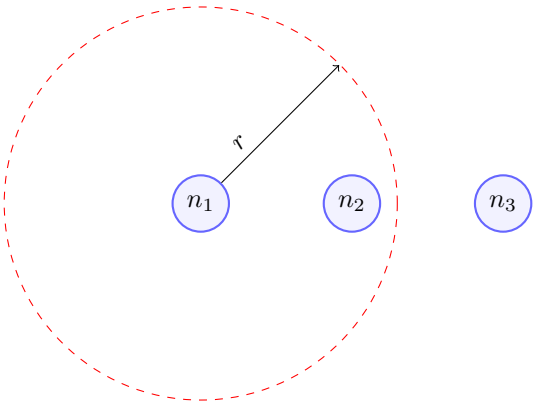

Fig. 1. Neighborhood

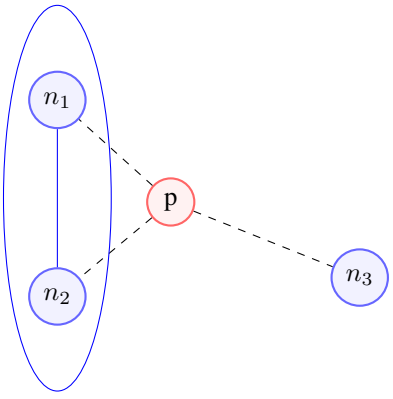

Fig. 2. Connection Rule

This rule attempts to encode the information of path connections between nodes by using the proximity of data points to nearby neurons. There is a guaranteed locality between two nodes relative to all other nodes. Angle information is not considered in the connection rule but could improve performance accuracy.

D. Collapsing

Due to this method's propensity to add more edges than necessary, the formation of false triangular subgraphs occurs. In this step, triangles are collapsed into a point.

All nodes which compose a triangle are listed and sorted by degree (number of edges). We iteratively collapse the triangles which contain the highest degree node. Suppose we have neurons n_1, n_2, n_3, n_4, n_5 in the following formation. We contract each edge into a single point whose weight is assigned the average of all nodes from the collapsed triangles.

$$n' = \frac{\sum_{i \in T} n_i}{|T|} \tag{6}$$

Where T is the set of all nodes in that collapsed triangle set.

E. Smoothing

After we have sufficiently contracted the network to remove triangular edges, the network goes through another weight adaption phase. Similar to the learning and growing phase, however, no new growth is allowed.

Algorithm 2 Smoothing

```
procedure SMOOTHING(N,D,r)
    r_large ← r * 2
    for 50 Runs do
        for p ∈ D do
            winner,error ← FindWinner(N,p)
            AdaptWeights(winner,p,r_large)
        end for
    end for
    r_small = r * 0.5
    for 50 Runs do
        for p ∈ D do
            winner,error ← FindWinner(N,p)
            AdaptWeights(winner,p,r_small)
        end for
    end for
end procedure
```

We double the initial neighborhood size to allow for newly contracted nodes to find a general area they should be in. We apply the learning phase for 50 iterations. We then set the neighborhood to half the original neighborhood size to precisely place every node to its most optimal location. This is applied to each data point for 20 iterations.

IV. Results & Comparison

While this method was designed for trail networks first, it is prudent to compare it to other methods. We compare our network to [4] Kernel Density Estimation (KDE) method and provided datasets. This dataset provides GPS traces from OSM in five different networks.

We construct and train a network for each given GPS trace to compare the most accurate methods. Then given ground truth geometry, we calculate the metrics: completeness (comp), precision, and topological correctness (topo) shown in Table **??**

A. Metrics

1) Segments: For the next two metrics and their respective algorithms, we make use of **segments**, which denotes s. Segments are defined as paths in a graph whose endpoints are nodes whose degree is either 1 or greater than 2.

Looking at the example in Fig. 3, we see a simple graph. On the right, we see the graph broken down into segments by color. Note orange nodes belong to each segment it is connected to. This simple graph has 6 total segments.

Additionally, each node in this graph is considered a shape point denoted in the following algorithms as $p \in s$.

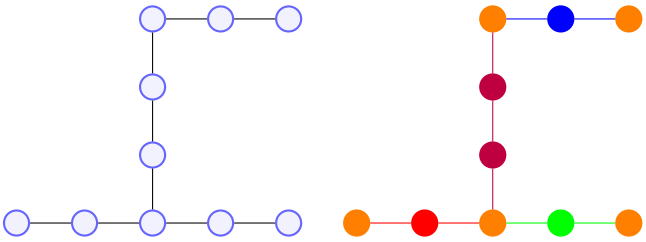

Fig. 3. Segments

2) Completeness: The completeness metric gives a percentage of how many constructed segments are matched to the ground truth network disregarding how accurate matched segments are. We have two parameters, T the ground truth network, and C the constructed network. We loop over segments $s_c \in C$ and $s_t \in T$. Essentially, for each segment in the constructed network, we find the closest segment in the ground truth network. Then we score how much total distance is matched by the constructed network so this value will always be between 0 and 1.

3) Precision: The precision metric gives the average distance in meters for matched segments as well as the standard deviation across all matched segments. Similar to completeness, we loop over each segment in the constructed network and find the closest segment in the ground truth network. We sum over the distances at each shape point along these segments and add to an overall precision array. Then we report the average distance over the entire array.

4) Topological Correctness: Topological correctness gives a quantitative measure of how connected the constructed network towards to the ground truth network. This is calculated by running the Floyd-Warshall algorithm on the constructed network's adjacency matrix A_c and ground truth adjacency matrix A_t.

The Floyd-Warshall algorithm finds all pairs' shortest path for a weighted graph [14]. In many implementations, this will be a matrix whose elements are the total path cost from one vertex to another. We then take an average of each of the returned matrices and produce the quotient.

Let A_t be adjacency matrix for ground truth network

Let A_c be adjacency matrix for constructed network

$F_t = \text{Floyd}(A_t)$

$F_c = \text{Floyd}(A_c)$

$a_t = \text{average of non-diagonal elements} F_t$

$a_c = \text{average of non-diagonal elements} F_c$

$$topo = \frac{a_t}{a_c}$$

B. Discussion

Fig. 4 presents the average performance for each network by our method and the provided data from Hashemi.[4] In each network, we have individual traces on which the algorithms were run and metrics produced.

Since each network has its own unique geographic features with varying sources, density, and total size, we compare mean performances across a network of traces.

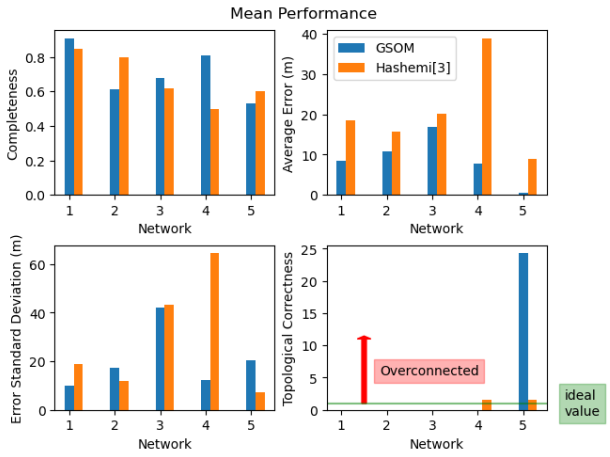

Fig. 4. Mean Metric Results Across Networks. Precision metric is shown as error.

Comparing our GSOM method to the KDE method from Hashemi, we see better performance overall on networks 1,3 and 4 with some improvements on other metrics for networks 2 and 5. Additionally, Hashemi's KDE method failed to construct a network on several traces in these networks, denoted by NA in Table **??** while our method constructs them but at a lower accuracy in many of the metrics than other traces from that network group. This will bring down the average performance for our method.

Overall, our network performs consistently better at completeness, average precision, and standard deviation precision. Our GSOM method does perform poorly at topological correctness with four of the networks resulting in 0. This means that there is at least one missed connection from the constructed network.

While the KDE method still has 3 networks with a topological correctness of 0, the other two have reasonable scores near 1 (the ideal value). While the one network, our method produces a non-zero score, it is an average of 24.32, meaning it produced too many connections.

V. Conclusion

In this paper, we presented a novel method for constructing trail and road networks with comparable performance to existing methods. We compare direct results on a variety of performance metrics used as a standard in this area. These metrics are relatively new, and historically most methods were published prior with a variety of non-standardized testing methods making it difficult to compare.

However, the results from this method are encouraging. We see improvements in traces from some networks compared to previous methods, such as KDE. In the original goal of constructing trail networks, this method improves upon existing methodology. With this in mind, the performance on road networks suggests this method could be expanded to include more dense road networks. Additionally, this algorithm can be applied to many public datasets with anonymized data, allowing for more crowd-sourced data.

References

[1] H. A. Karimi and P. Kasemsuppakorn, "Pedestrian network map generation approaches and recommendation," *International Journal of Geographical Information Science*, vol. 27, no. 5, p. 947–962, 2013.

[2] G. Kumar, P. K. Kalra, and S. G. Dhande, "Curve and surface reconstruction from points: An approach based on self-organizing maps," *Applied Soft Computing*, vol. 5, no. 1, p. 55–66, 2004.

[3] L. Yun and K. Uchimura, "Using self-organizing map for road network extraction from ikonos imagery," *International Journal of Innovative Computing, Information and Control*, p. 641–656, 2006.

[4] M. Hashemi, "A testbed for evaluating network construction algorithms from gps traces," *Computers, Environment and Urban Systems*, vol. 66, p. 96–109, 2017.

[5] P. P. Singh and R. D. Garg, "Automatic road extraction from high resolution satellite image using adaptive global thresholding and morphological operations," *Journal of the Indian Society of Remote Sensing*, vol. 41, no. 3, p. 631–640, 2012.

[6] C. Tao, J. Qi, Y. Li, H. Wang, and H. Li, "Spatial information inference net: Road extraction using road-specific contextual information," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 158, p. 155–166, 2019.

[7] P. Chao, W. Hua, R. Mao, J. Xu, and X. Zhou, "A survey and quantitative study on map inference algorithms from gps trajectories," *IEEE Transactions on Knowledge and Data Engineering*, p. 1–1, 2020.

[8] X. Liu, J. Biagioni, J. Eriksson, Y. Wang, G. Forman, and Y. Zhu, "Mining large-scale, sparse gps traces for map inference," *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012.

[9] R. Stanojevic, S. Thirumuruganathan, S. Chawla, F. Filali, and A. Aleimat, "Kharita: Robust map inference using graph spanners," *arXiv*, Feb 2017.

[10] X. Yang, L. Tang, C. Ren, Y. Chen, Z. Xie, and Q. Li, "Pedestrian network generation based on crowdsourced tracking data," *International Journal of Geographical Information Science*, vol. 34, no. 5, p. 1051–1074, 2019.

[11] S. Karagiorgou and D. Pfoser, "On vehicle tracking data-based road network generation," *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, 2012.

[12] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, p. 509–517, 1975.

[13] S. Marsland, *Machine learning: An algorithmic perspective*. CRC Press, 2011.

[14] R. W. Floyd, "Algorithm 97: Shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962.