An Efficient Algorithm for the Construction of Dynamically Updating Trajectory Networks

Deniz Gurevin
Electrical and Computer Engineering
University of Connecticut, CT USA
deniz.gurevin@uconn.edu

Chris J. Michael
U.S. Naval Research Laboratory
Stennis Space Center, MS USA
chris.michael@nrlssc.navy.mil

Omer Khan
Electrical and Computer Engineering
University of Connecticut, CT USA
khan@uconn.edu

***Abstract*—Trajectory based spatiotemporal networks (STN) are useful in a wide range of applications, such as crowd behavior analysis. Significant portion of trajectory network based research focuses on optimizing the analysis of STN to characterize, control, and predict network behavior. However, these mining algorithms are typically carried out on a pre-constructed network structure that tracks all moving objects and their trajectories in real time. The construction of such a trajectory network is itself a computationally expensive task and it is becoming a bigger burden with advancements in analysis algorithms. The traditional approach is to construct static networks from the temporal snapshots of trajectory data that cannot handle spatiotemporally changing data. This paper proposes an efficient algorithm that successfully generates and maintains an ST network from raw trajectory data. The proposed method is based on a customized R-Tree based constructor to keep track of object trajectories and their interactions. It avoids redundant updates as trajectories evolve over time, resulting in a significant reduction in STN construction time. Based on the experiments we conducted, our method reduces the number of updates by 71.25% as compared to the static naive STN construction method.**

***Index Terms*—trajectory networks, trajectory data mining, spatiotemporal networks, R-Tree, network construction**

I. Introduction

In recent years, there has been a rapid development in application areas of satellite systems, vehicle navigation, motion planning and location acquisition that have all led to the growth of massive databases containing trajectory data of moving objects. Analysis of spatiotemporal networks (STN) consisting of trajectory data is used to characterize, control, and predict network behavior that can be useful in a broad range of applications, such as analysis of transportation systems, GPS-based systems and crowd behavior analysis [1]–[5]. For this reason, the detection and mining of important nodes using STNs (where nodes possess spatiality) have become a popular problem in data science. Depending on the system and application, the concept of node importance may differ. This can include calculating the node that is closest to the rest of the nodes of the network (closeness centrality [6]), most connected node (degree centrality [7]), node connected to other important nodes (eigenvector centrality [8]), node which passes most information (closeness centrality [9]), etc.

The algorithms that calculate these metrics are often designed to execute on a pre-constructed STN consisting of raw trajectory data. Therefore, a trajectory network is initially constructed, which results in a graph representation. Here, graph nodes represent the objects, and edges represent the relationships between objects that are in close proximity to each other, forming *contacts*. Current research on node importance generally focuses on the optimization of mining algorithms and aims to achieve faster solutions [10]–[13]. In general, node importance mining algorithms can be exact [14]–[16], i.e. they will always accurately compute the final values of the metrics, or they can be based on approximation to speed up performance since the exact values of metrics can be prohibitive for large graph computations in practice [13], [17]. Regardless of their approach, all of these mining algorithms require a pre-constructed trajectory network to work on. However, with the advancements in network analysis algorithms, the construction of the trajectory network itself becomes a computationally expensive task that requires considerable time. This step is generally neglected in current research.

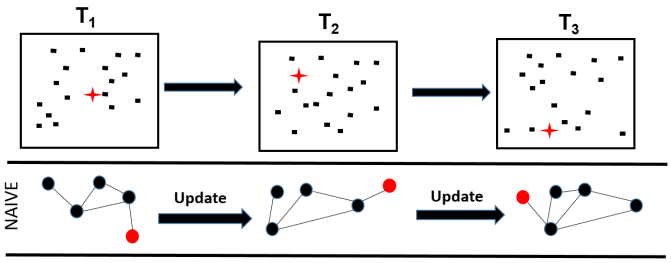


Fig. 1: Naive approach to handle spatiotemporally changing data.

A traditional naive approach to address this problem is shown in Figure 1. T_1, T_2 and T_3 represent the sequential timestamps that contain moving objects on the Euclidean plane. Objects can form contacts with other objects on the plane if they remain in close proximity with each other. The proximity rule is a predefined metric and can be defined as the maximum distance the objects need to maintain between each other in order to keep their contact. Since objects may change their position at different timestamps, the contacts they form may change. The naive trajectory construction method constructs static networks from the snapshots of timestamps, and cannot handle the temporal updates in an efficient way. In fact, this type of approach has to assign many redundant updates even if the underlying graph does not change over time.

This paper focuses on spatiotemporal trajectory network

generation from trajectory data, based on a predefined proximity rule. We investigate the existing challenges in the network generation and identify the computational cost for network construction depending on the characteristics of the trajectory data. Based on the observations, we propose a new method that efficiently constructs a proximity based graph representation. First, trajectories are processed and the objects that form contacts with each other are clustered and structured in a customized R-Tree based representation at any given initial timestamp. As trajectories change in real time, only necessary nodes are updated at the leaf level of the representative structure. This way, the algorithm avoids redundant updates and computations. Node mining algorithms then perform their computations by extracting the necessary graph representation from the tree structure. We evaluate a variety of cases and scenarios where dataset parameters change, such as the speed of the objects and their proximity threshold as well as area size, the number of objects within the area and the number of timestamps to be processed. Based on the experiments on real and synthetic datasets, our construction/update method reduces the number of updates by 71.25% as compared to the naive method. This leads to fast processing and updates of trajectory network that accelerates mining algorithm computations in real time.

II. Background

In the context of trajectories, a graph or network can represent the potential interaction of objects. A trajectory network or graph, denoted as $G[t,T]$ with an observation time interval $[t,T]$ is made up from a set of vertices $V[t,T]$ that are connected by a set of edges $E[t,T]$. $V[t,T]$ represents all moving objects N in a Euclidean plane, while $E[t,T]$ represents the links between nodes in time. In order to maintain and process temporally changing data, the network is processed into a sequence of discrete $T-t$ network "snapshots". Therefore, a temporal network is traditionally represented as a sequence of static graphs $(G_1, G_2, ..., G_{T-t})$.

In order to represent the physical proximity between objects (also called *contacts*) at time t, the distances between all pairs of objects $\in V_t$ are calculated. If the distance is less than or equal to the predefined proximity threshold τ, then this is considered a link or an event and is added to E_t. This process is repeated until all events between objects are found. The computational cost of this process is $O(T \cdot |V_t|^2)$ for $[0,T]$. This type of traditional naive construction has the following drawbacks:

- It is not a dynamic structure. It has to construct a new static network from scratch at each timestamp, making it computationally expensive.
- It is likely to contain excessive amounts of redundant data. Even if objects do not change their location from one timestamp to the next, it still examines each object at each timestamp and constructs the same network.

In terms of mining trajectory networks, any dataset can be represented using a set of nodes and a set of links (edges) that characterize the relationship between them. Depending on the system/application, the concept of node importance may differ. Several methods have been studied and proposed to mine trajectory data and the concept of node importance. [12] focuses on comparing the similarity of movement characteristics of moving objects. Algorithms that identify similarity in trajectory data [10], [11] and trajectory clustering [5] can be used in grouping similar trajectories. In terms of spatial networks, there has been research that explored the objects distributed in space and interacting with each other such as proximity graphs [18] and geometric intersection graphs [19]. These proximity graphs include relative neighbor graphs [20] and Gabriel graphs [18] that connect nearest neighbors and Delaunay triangulations which maximize the minimum angles of all triangles formed. In networks evolving over time, with the addition of the temporal information, new metrics can be adopted such as temporal node centrality [21] and network reachability/connectedness [22].

The Sweep Line Over Trajectories (SLOT) algorithm [13] is a method that can simultaneously evaluate node importance metrics for all moving objects in the trajectory network. The SLOT algorithm avoids a large number of unnecessary computations by evaluating the network and updating the metrics of interest only when an event between two nodes is formed and dissolved (start and end time of an event), instead of evaluating the network at each snapshot. For the details of the SLOT algorithm, we refer the readers to [13]. Given the trajectories of N objects, an observation time interval $[t,T]$, and a proximity threshold τ that defines a contact between two objects, SLOT computes the metrics that define:

- the trajectory node degree of each object.
- the trajectory node connectedness of each object.
- the trajectory node triangle membership of each object.

In this paper, we use the SLOT algorithm to calculate the aforementioned node importance metrics. Note that, our network construction algorithm performance is not dependent on SLOT algorithm. We use an algorithm for node importance metrics only for the purpose of demonstrating the performance of our strategy along with the overhead of node importance calculations. In fact, any node importance mining algorithm can be used with our construction method. We choose SLOT algorithm because (1) SLOT specifically focuses on interactions of nodes, (2) it can handle spatiotemporal data, (3) it is an exact algorithm, therefore, it always captures the correct values of the metrics of interest and provides a detailed analysis of node importance, and (4) it is a fast algorithm that handles node importance computations on large scale datasets.

III. Trajectory Network Construction using R-Tree based Structure

Instead of generating a static network from scratch at each timestamp, the proposed method creates a customized R-Tree based structure and uses a dynamic updating algorithm that enables the network structure to support spatiotemporally changing data. Many variants of the R-Tree have been developed [23] since the original structure was proposed in [24]. We will use an R-Tree based structure to keep track of all objects in the trajectory network, and use a customized update

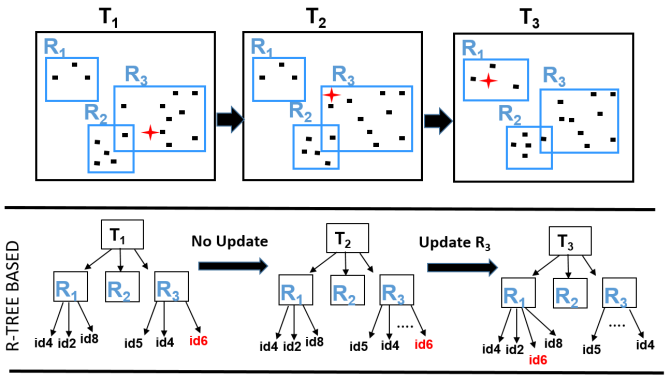


Fig. 2: Trajectory network construction method clusters objects that have a distance $\leq \tau$ with each other. These clusters or *MBRs* are represented in a tree structure. While many objects change their trajectories in the next timestamps, no updates are needed in the tree structure as long as the objects remain in their current MBR.

algorithm to update the network in the following timestamps. However, the comparison of our proposed method against other R-tree implementations, which were discussed in [24], is left for future work.

Consider a set of N objects in $\mathbb{R}^2$ on which we want to build an index structure. Rather than indexing the object themselves, each object is represented by its minimum bounding rectangle or MBR. Thus, the objects are clustered in rectangles in $\mathbb{R}^2$, that is represented by its corner coordinates $MBR_{(x_1,y_1),(x_2,y_2)}$. In the case of trajectory networks, MBRs are considered as *clusters that group all objects on the Euclidean plane that form contacts with each other*, i.e. each object pair within the MBR maintains a distance less than or equal to the predefined threshold τ. MBR corner coordinates $(x_1, y_1), (x_2, y_2)$ are updated accordingly.

Figure 2 shows an example of how the proposed trajectory network construction method works. It shows timestamps T_1, T_2 and T_3 that represent snapshots of the trajectories taken chronologically. All objects are clustered to form events with each other (distance $\leq \tau$) in MBRs $R_1, R_2, R_3, ..., R_M$ and form an R-Tree based structure that maintains the MBRs and the objects inside them at the leaf level. When moved from T_1 to T_2, it can be seen that some objects do change their location; however, the change in the coordinates is not significant since all objects remain within their MBR coordinates, i.e. objects maintain their current contacts. In this case, an update at the tree nodes is not required. However, at T_3, an object leaves its MBR R_3 and moves to R_1. In this case, an update to the R-Tree at the third leaf level is required. Object ID is removed from the R_3 node and added to R_1. Of course, the MBR corner coordinates need to be updated simultaneously.

Initially, given trajectory data, the first step of the algorithm is to construct a network from scratch. This process is explained in Algorithm 1. All trajectories at an initial time t are first collected (Line 3). The algorithm then performs the following for each object in the current trajectory data: it iterates

Algorithm 1 Trajectory Network Construction

Given the trajectory data that contains the timestamp $t \in \{0, T-1\}$, object id id and object coordinates (x, y), the algorithm constructs an R-Tree based structure and its corresponding MBRs.

1: **procedure** CONSTRUCTRTREE(data)
2: $t = initialtimestamp$
3: $trajectories_t \leftarrow$ extractData(t)
4: **for** $object$ in $trajectories_t$ **do**
5: **for** mbr in $MBRList$ **do**
6: **if** $isInside((x, y), mbr_{coor}) == True$ **then**
7: $mbr.add(object)$
8: $mbr.updateCorners()$
9: $objLookup[id].add(mbr_{id})$
10: **end if**
11: **end for**
12: **if** $length(objLookup[id]) == 0$ **then**
13: $mbrNew = createMBR()$
14: $mbrNew.add(object)$
15: $mbrNew.updateCorners()$
16: $objLookup[id].add(mbrNew_{id})$
17: **end if**
18: **end for**
19: **end procedure**

Algorithm 2 Trajectory Network Update

Given the trajectory data at timestamp $t \in \{1, 2, ...T\}$, object id id and object coordinates (x, y), the algorithm compares the objects movements with the previous timestamps and assigns the necessary updates.

1: **procedure** UPDATERTREE($data, t$)
2: $trajectories_t \leftarrow$ extractData(t)
3: **for** $object$ in $objLookup[id]$ **do**
4: **for** mbr in $MBRList$ **do**
5: $inside = isInside((x, y), mbr_{coor})$
6: **if** mbr_{id} in $objLookup[id]$ **then**
7: **if** $inside == True$ **then**
8: $mbr.updateCorners()$
9: **else**
10: $mbr.remove(object)$
11: $mbr.updateCorners()$
12: $objLookup[id].remove(mbr_{id})$
13: **end if**
14: **else**
15: **if** $inside == True$ **then**
16: $mbr.add(object)$
17: $mbr.updateCorners()$
18: $objLookup[id].add(mbr_{id})$
19: **end if**
20: **end if**
21: **if** $length(objLookup[id]) == 0$ **then**
22: $mbrNew = createMBR()$
23: $mbrNew.add(object)$
24: $mbrNew.updateCorners()$
25: $objLookup[id].add(mbrNew_{id})$
26: **end if**
27: **end for**
28: **end for**
29: **end procedure**

through each MBR in the tree nodes, denoted by $MBRList$ and checks whether the object can fit inside the MBR using the $isInside(.)$ procedure (Lines 4–6). Given object coordinates (x, y) and MBR coordinates mbr_{coor}, $isInside(.)$ procedure checks if object coordinates are within the boundaries of mbr_{coor}. If it is, it returns $True$. If the object is not inside the MBR, it might be the case that object is still within the threshold of MBR boundaries and MBR is expanded to fit the object. In order to check this, $isInside(.)$ procedure

calculates the distance between object coordinates and MBR corners and checks whether $d = (d_1, d_2, d_3, d_4) \leq \tau$, i.e., the object can fit into the boundary of the MBR. If it satisfies this condition, it returns $True$. Based on this result, the object ID is added to the MBR and corner coordinates are updated using $updateCorners()$. As a result, this expands or shrinks the boundaries of MBR based on the coordinates of the outermost objects that are the members of the MBR. Also, a separate object lookup table is maintained, which keeps the information about objects and their corresponding MBRs. If the object is added to the MBR, then the object lookup table is updated with that particular MBR's ID in the index of the object (Lines 7–9). At the end of the search for MBRs, the object lookup table for the object's index is checked. If it is still empty, the object is currently not a member of any MBR. Then, a new MBR for the object itself is created, and the rest of the updates take place (Lines 12–17). The iterations are repeated until all objects find the MBRs they can fit in.

Once, the initial R-Tree based network structure is constructed, at subsequent timestamps, the structure is no longer constructed from scratch but updated depending on the object movements. This procedure is explained in Algorithm 2. The algorithm does the following for each object in the object lookup table: it iterates through the current MBRs and calls the $isInside(.)$ procedure and checks whether the object's coordinates can fit inside the MBR (Line 3–5). Then, it checks whether the object is currently a member of the corresponding MBR using the object lookup table. If the object is already a member of the MBR, and if the object's newest location still lies within that MBR, only the MBR coordinates need to be updated. If it is outside the MBR, the object is removed from the MBR, and updates are performed on the MBR's coordinates and object lookup table (Lines 6–13). On the other hand, if the object is not already a member of the MBR and if it can fit inside the MBR, the object ID is added to the MBR (Lines 14–19). At the end of the updates, the object lookup table for the object's index is again checked. If the object is currently not a member of any MBR, a new MBR for the object itself is created (Lines 21–26). Again, this process is repeated until all objects find their corresponding MBRs. At the end of this process, MBRs contain all objects that form events with each other.

After the tree structure is updated at a given timestamp t, the events are processed using Algorithm 3. Each MBR in the tree and its entries are examined (Lines 3–5). Nodes V_t are updated with the objects that lie within the MBR borders. Each object pair inside the MBR is then added to E_t since they form events with each other (Lines 6–8). At this point, the mining algorithms can be executed for the events at the given timestamp. This process is repeated as the trajectory network evolves spatiotemporally.

Algorithm 3 Event Update

Given the a R-Tree based trajectory network at timestamp $t \in \{1, 2, ...T\}$, the algorithm processes the sequence of edge events.

```
1: procedure PROCESSEVENTS(RTree,t)
2:     edges = [], nodes = []
3:     mbrList ← RTree(t)
4:     for mbr in mbrList do
5:         for i in range(0, length(mbr.objects) do
6:             nodes.append(mbr.objects[i])
7:             for j in range(i + 1, length(mbr.objects) do
8:                 edges.append(mbr.objects[i], mbr.objects[j])
9:             end for
10:        end for
11:    end for
12:    V[t].append(nodes)
13:    E[t].append(edges)
14: end procedure
```

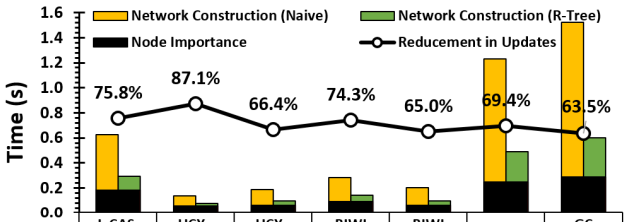
IV. Experimental Evaluation

In this Section, the proposed trajectory construction method is evaluated and compared with the naive method. We show the overhead of both approaches when run with the node importance SLOT mining algorithms [13]. We also run experiments to further understand and evaluate the effect of some critical parameters on the computation cost, such as speed of trajectories, the number of objects (N), the number of timestamps (T), the value of threshold (τ). We initially provide details of the computational environment and the datasets.

Environment: All experiments are conducted on an Intel(R) Core(TM) i7-7820HQ CPU @ $2.90GHz$ and $32GB$ memory. The system is booted with Windows 10 operating system. The Python 3.6 programming language is used for the trajectory network construction and node importance algorithms.

Data: In order to test the proposed method's performance, publicly available datasets in trajectory networks are used and the parameters of them are fixed in our experiments as following: ETH Walking Pedestrians Dataset [25] (BIWI ETH with $N = 360$, $T = 1448$ and 8908 trajectories, and BIWI Hotel with $N = 405$, $T = 1169$ and 6544 trajectories), UCY (Crowds-by-Example) Dataset [26] (UCY Zara-1 with $N = 148$, $T = 866$ and 3882 trajectories, and UCY University-1 with $N = 415$, $T = 444$ and 5779 trajectories), L-CAS 3DOF Pedestrian Trajectory Dataset [27] ($N = 631$, $T = 1700$ and 21887 trajectories), Stanford Drone Dataset [28] ($N = 1186$, $T = 2000$ and 44532 trajectories), and Grand Central Station Dataset [29], [30] ($N = 2500$, $T = 1400$ and 47866 trajectories). Size of each trajectory entry is 0.12KB.

We also generate synthetic trajectory data to evaluate and understand the effects of various cases and data parameters on our algorithm. Two types of synthetic data generators are used. First, the generator for random velocity trajectories over a Euclidean plane from [13] is adopted. Given the number of timestamps T and the number of objects N, for a period of $[0, T]$, the generator generates random trajectories. The details of these parameters are given as we go through experiments. However, we fix certain parameters throughout the experimental section, such as space size = 1000×1000, minimum speed = 0 and maximum speed = 1. Secondly, the idea of Brownian motion is adopted to generate more realistic random pedestrian trajectories. For this generator, the space size is fixed to 100×100. The details of the other parameters are given throughout the section. Using these synthetic datasets,



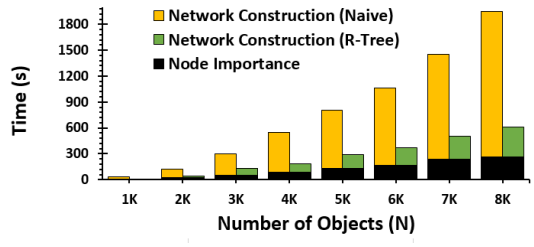
	L-CAS 3DOF	UCY - Zara	UCY - Univ.	BIWI ETH	BIWI Hotel	Stanford	GC Station
Updates (R-Tree)	210.3	35.1	118.4	129.2	187.2	331.4	666.1
Updates (Naive)	869.1	271.3	352.1	502.4	534.1	1084.2	1824.9

Fig. 3: Proposed trajectory network construction method's performance with node importance algorithms on different non-synthetic datasets. When compared with node importance algorithms, it can be observed that the overhead of our construction algorithm is less on smaller datasets.

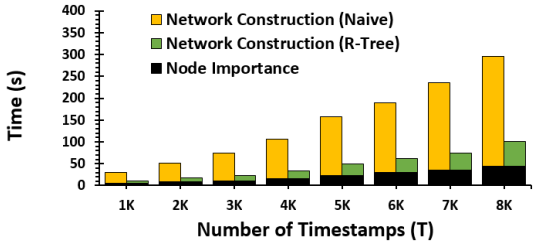
the dataset parameters are varied, and their effects are observed on the algorithm performance.

We start by evaluating the performance of trajectory network construction together with the three node importance SLOT algorithms (c.f. Section II). As previously stated, for any node importance algorithm, the data first needs to be collected and the network needs to be constructed or updated with the new data. Until after the network or graph becomes ready, the node importance algorithms cannot be executed. The experiment shown in Figure 3 observes the overhead of trajectory network construction on the node importance algorithm on various real-world datasets. The average number of updates to an object per timestamp is also shown for both methods. In the context of R-Tree based network construction, update means removing/adding an object from/to an MBR. Since the naive method does not maintain such a structure, update means processing an event for the object. Based on our experiments, the node importance algorithm is a lot faster than the naive network construction. It can be observed that the performance of both methods is highly correlated with the number of updates, which happens to be a lot fewer for our proposed method. This is because our algorithm does not need to assign an update to an object at each timestamp, but only when it is necessary (e.g. once an object leaves its MBR). The naive method, however, does not evaluate this redundancy and assigns updates to each object at each timestamp even if the objects do not change their position. Based on these experiments, the proposed method reduces the average number of updates per timestamp by a geometric mean of 71.25%.

Similarly, the experiment shown in Figure 4 observes the overhead of trajectory network construction on the node importance algorithm on random synthetic data. In Figure 4a, the number of timestamps are fixed to $T = 10^3$, and the number of objects are increased from 10^3 to 8×10^3. In Figure 4b, the number of objects are fixed to $N = 10^3$ and the number of timestamps are increased from 10^3 to 8×10^3. In both cases, we observe the time required to run the node importance together with the naive method for trajectory network construction versus node importance together with our proposed algorithm for network construction. At $T = 10^3$ and $N = 8 \times 10^3$,



(a) $T = 1000$.



(b) $N = 1000$.

Fig. 4: Trajectory network construction methods when run together with node importance algorithms. Construction of the network with the naive approach takes even longer than the node importance algorithms.

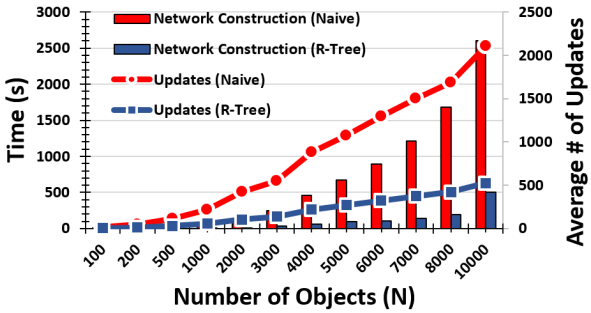


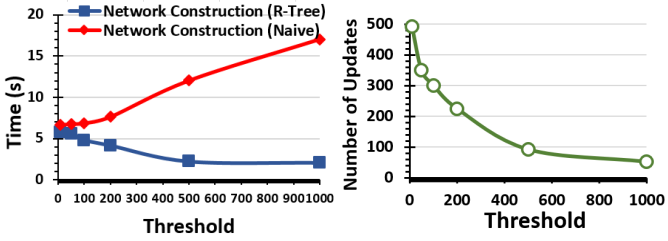
Fig. 5: Trajectory network construction performance with different values of N and average number of updates per timestamp on synthetically generated Brownian motion data.

it can be seen that the network construction takes more than $6\times$ longer than the node importance algorithms in the naive method's case. This becomes problematic in cases where the node importance algorithms and network construction are pipelined. In such a case, the pipeline granularity will degrade since the wait time for the node importance algorithm increases exponentially, especially on large scale data and an increased number of objects. However, we show that the proposed network construction method reduces the overhead as close as to the node importance runtime. This makes our algorithm ideal for cases where the node importance and network construction are pipelined.

Additionally, it can be observed that both algorithms are more sensitive to N compared with T. This is because the interaction of objects with each other needs to be taken into account for the node importance algorithms. However, it can be observed that the proposed construction method is more

robust against large N compared to the naive method. The naive construction method needs to examine each object pair to evaluate their interaction (in this case, calculate the distance between them, and therefore is $\mathcal{O}(N^2)$. On the other hand, our network construction does not need to evaluate each object pair. It instead examines the interactions between objects and MBRs, which makes it $\mathcal{O}(N \cdot M)$, where M is the number of MBRs which is much smaller than N.

Figure 5 shows the experiment performed on the synthetic Brownian motion data. The number of timestamps is fixed to $T = 10^3$, and the number of objects is increased from 10^2 to 10^4. It additionally shows the average number of updates to an object per timestamp. The number of assigned updates increase with the increased N. However, since the proposed network construction takes advantage of the redundant data, it assigns not as many updates than the naive method.



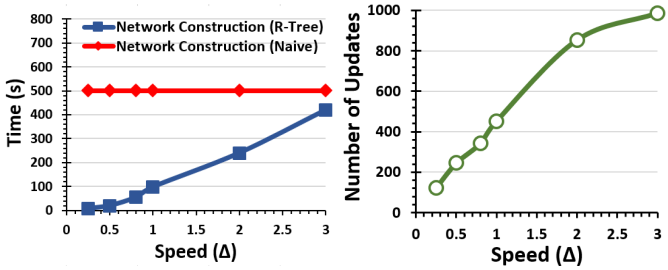
(a) Threshold effect on the methods.

(b) Average number of updates per timestamp in the proposed construction method.

Fig. 6: Comparison of the trajectory network construction methods with respect to different thresholds on the synthetic random trajectory data (Figure 6a). Unlike the naive method, the proposed construction method gets faster as threshold increases, since fewer updates are assigned (Figure 6b).

Figure 6 shows the effect of distance threshold (τ) on the algorithms on the synthetic random trajectory data. As previously mentioned, the Euclidean plane size here is 1000×1000. The experiment sets $N = 5 \times 10^2$ and $T = 10^3$. In Figure 6a, τ is increased from 10 to 10^3. We observe that as τ increases, the performance of our construction method improves while the performance of the naive construction algorithm degrades. This opposing behavior is explained with the number of MBRs (M) maintained by the R-Tree structure. When threshold τ increases, it allows for more objects on the plane to interact with each other, i.e. it increases the number of events and edges in the network. In this case, the naive method has to process more events. In R-Tree structure, each MBR contains objects that form events with each other. When the threshold τ is high (proportioned with the area size), there are a smaller number of MBRs in the R-Tree that contain more objects and fewer updates are required. When the threshold τ decreases, R-Tree needs to generate more MBRs that contain fewer objects and assign more updates as shown in Figure 6b. The work in R-Tree increases with the number of MBRs it needs to maintain. Therefore, the nature of R-Tree allows our proposed construction method to handle higher τ better.

Figure 7 shows the experiment on the synthetic Brownian motion trajectory data with parameters $N = 1000$, $T = 1000$



(a) Effects of object speed.

(b) Avg. updates per timestamp in our construction method.

Fig. 7: Our trajectory network construction performance with respect to different Δ values as compared to the naive method on the Brownian Motion data (Figure 7a). As Δ decreases, objects move at lower speeds and generate more redundant data; therefore, smaller number of updates to the network need to be made (Figure 7b).

and $\tau = 5$. The speed of the objects is changed using the Δ parameter in the Brownian motion equation. When the speed of moving objects is low, as timestamps are processed, objects do not change their position as frequently. Therefore, they generate a lot of redundant information. For example, if the speed is low, the objects in the network may not leave their MBRs at all or may leave after many timestamps. Therefore, no updates need to take place for a certain length of observation time. Even if the objects change their position to some extent, as long as they remain in their current MBRs, no updates may be required. As shown in Figure 7b, the number of updates in the trajectory network increase with the speed. Therefore, in Figure 7a, we observe that as the speed increases (i.e. the redundancy decreases), our algorithm's performance degrades relative to the naive method.

V. Discussion and Conclusion

A novel trajectory network construction algorithm using R-Tree based structure is proposed that efficiently generates an STN from raw trajectory data. We show that the method can significantly reduce the time cost of the network construction. It keeps track of all trajectories and their interactions using Minimum Bounding Rectangles (MBR), and therefore, it efficiently handles spatiotemporally changing data. The method is evaluated against the static network construction method using various real-world and synthetic datasets. It is shown that the proposed method significantly outperforms static network construction, especially in highly redundant datasets. Additionally, it is shown to reduce data construction time to as low as the time required to execute the node importance algorithms on a trajectory network. This enables fast trajectory network computations since the network construction can be pipelined with mining algorithms. The proposed method can be further optimized since the R-Tree structure used in this paper has a depth of 1, and no limitations are imposed on the number of objects per MBR. A generalized R-Tree constructor can support arbitrary depth and perform self-balancing operations that can be utilized for future optimizations.

ACKNOWLEDGEMENTS

This work was funded by the U.S. Government under a grant by the Naval Research Laboratory. This work was also supported by the National Science Foundation (NSF) under Grant No. CNS-1718481.

REFERENCES

[1] K. Sila-Nowicka, J. Vandrol, T. Oshan, J. Long, U. Demsar, and A. Fotheringham, "Analysis of human mobility patterns from gps trajectories and contextual information," *International Journal of Geographical Information Science*, vol. 30, pp. 881 – 906, 2016.

[2] A. Sawas, A. Abuolaim, M. Afifi, and M. Papagelis, "Tensor methods for group pattern discovery of pedestrian trajectories," *2018 19th IEEE International Conference on Mobile Data Management (MDM)*, pp. 76–85, 2018.

[3] D. Guo, S. Liu, and H. Jin, "A graph-based approach to vehicle trajectory analysis," *Journal of Location Based Services*, vol. 4, pp. 183 – 199, 2010.

[4] J. Kim, K. Zheng, J. Corcoran, S. Ahn, and M. Papamanolis, "Trajectory flow map: Graph-based approach to analysing temporal evolution of aggregated traffic flows in large-scale urban networks," 2017.

[5] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *SIGMOD '07*, 2007.

[6] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, pp. 581–603, 1966.

[7] S. Wasserman and K. Faust, "Social network analysis - methods and applications," in *Structural analysis in the social sciences*, 2007.

[8] P. Bonacich, "Power and centrality: A family of measures," *American Journal of Sociology*, vol. 92, pp. 1170 – 1182, 1987.

[9] U. Brandes, "A faster algorithm for betweenness centrality," *The Journal of Mathematical Sociology*, vol. 25, pp. 163 – 177, 2001.

[10] M. V. Kreveld and J. Luo, "The definition and computation of trajectory and subtrajectory similarity," in *GIS*, 2007.

[11] N. Magdy, M. Sakr, T. Mostafa, and K. El-Bahnasy, "Review on trajectory similarity measures," *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS)*, pp. 613–619, 2015.

[12] S. Dodge, R. Weibel, and E. Forootan, "Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects," *Comput. Environ. Urban Syst.*, vol. 33, pp. 419–434, 2009.

[13] T. Pechlivanoglou and M. Papagelis, "Fast and accurate mining of node importance in trajectory networks," *2018 IEEE International Conference on Big Data (Big Data)*, pp. 781–790, 2018.

[14] L. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting triangles in data streams," in *PODS '06*, 2006.

[15] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, "Reductions in streaming algorithms, with an application to counting triangles in graphs," in *SODA '02*, 2002.

[16] C. E. Tsourakakis, U. Kang, G. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *KDD*, 2009.

[17] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017.

[18] K. Gabriel and R. Sokal, "A new statistical approach to geographic variation analysis," *Systematic Biology*, vol. 18, pp. 259–278, 1969.

[19] T. Erlebach, K. Jansen, and E. Seidel, "Polynomial-time approximation schemes for geometric intersection graphs," *SIAM J. Comput.*, vol. 34, pp. 1302–1323, 2005.

[20] G. Toussaint, "The relative neighbourhood graph of a finite planar set," *Pattern Recognit.*, vol. 12, pp. 261–268, 1980.

[21] H. Kim and R. Anderson, "Temporal node centrality in complex networks." *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 85 2 Pt 2, p. 026107, 2012.

[22] P. Holme, "Network reachability of real-world contact sequences." *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 71 4 Pt 2, p. 046119, 2005.

[23] L.-V. Nguyen-Dinh, W. G. Aref, and M. Mokbel, "Spatio-temporal access methods: Part 2 (2003 - 2010)," *IEEE Data Eng. Bull.*, vol. 33, pp. 46–55, 2010.

[24] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *SIGMOD '84*, 1984.

[25] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 261–268.

[26] A. Lerner, Y. Chrysanthou, and D. Lischinski, "Crowds by example," in *Computer graphics forum*, vol. 26, no. 3. Wiley Online Library, 2007, pp. 655–664.

[27] L. Sun, Z. Yan, S. M. Mellado, M. Hanheide, and T. Duckett, "3dof pedestrian trajectory prediction learned from long-term autonomous mobile robot deployment data," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–7, 2018.

[28] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, "Learning social etiquette: Human trajectory understanding in crowded scenes," in *European conference on computer vision*. Springer, 2016, pp. 549–565.

[29] B. Zhou, X. Wang, and X. Tang, "Random field topic model for semantic region analysis in crowded scenes from tracklets," in *CVPR 2011*. IEEE, 2011, pp. 3441–3448.

[30] S. Yi, H. Li, and X. Wang, "Understanding pedestrian behaviors from stationary crowd groups," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3488–3496.