Fast and Accurate Mining of Node Importance in Trajectory Networks

Tilemachos Pechlivanoglou and Manos Papagelis
Lassonde School of Engineering, York University, Toronto, Canada
tipech@eecs.yorku.ca, papaggel@eecs.yorku.ca

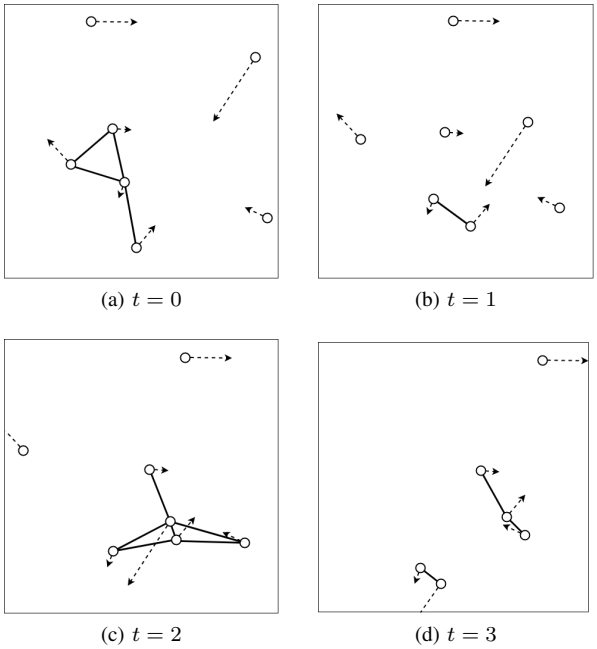
***Abstract*—Mining large-scale trajectory data streams (of moving objects) has attracted significant attention due to an abundance of modern tracking devices and a number of real-world applications. In this paper, we are interested in evaluating the relative importance of such objects through monitoring their interactions with other objects, over time. Which object has encountered more other objects? When did these encounters happen and how long did they last? To address this type of questions, we consider a trajectory network that is defined based on the proximity of moving objects over time. Given this network, we are able to evaluate the importance of an object (node) by monitoring its complex network connections to other nodes over time. Traditional approaches to address the problem rely on either evaluating network metrics over a number of static network snapshots or expensive trajectory similarity and clustering methods that require further post-processing. Streaming algorithms also exist, but they focus on simple network metrics. In contrast to these approaches, we devise a method that is able to simultaneously evaluate node importance metrics for all moving objects in the trajectory network. Our proposed method is based on, first, efficiently computing and representing the interactions of moving objects as time intervals. Then, a fast and accurate one-pass sweep-line algorithm over the trajectories (SLOT) is devised that can effectively compute the metrics of interest, all at once. Through experiments on various types of data, we demonstrate that our algorithm is a multitude of times faster than sensible baselines, for a varying range of conditions.**

***Index Terms*—Trajectory data mining, graph mining, network science, trajectory networks, dynamic networks**

I. Introduction

Advances in location acquisition and tracking devices have given rise to the generation of enormous trajectory data consisting of *spatial* and *temporal* information of moving objects, such as persons, vehicles or animals [1]. Mining trajectory data to find interesting patterns is of increased research interest due to a broad range of useful applications, including analysis of transportation systems, location-based services, and crowd behavior analysis [2]–[5].

As multiple objects are continuously moving in an area they can be found in close proximity to each other, forming *contacts*. In this paper, proximity means spatial distance. A *contact* between two objects can be seen as an *event* that lasts for as much as their spatial distance remains consistently smaller than a proximity threshold τ. Therefore, any event has a duration, simply defined as the amount of time that passes from the beginning of the contact until it ends. We say that a contact is *active* for the duration of the event describing it. Note that any two objects might encounter each other multiple times, leading to the same contact occurring multiple times. So, while each event is unique, multiple events might refer to the same contact of two objects, occurring at different times.



(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$

Fig. 1. Example evolution of the system over discrete times. As objects move over trajectories they form contacts with other objects they are in close proximity to. The collection of contacts at time t represents a proximity network; over a period of time $[0, T]$ it represents a temporal network, or *trajectory network*. We want to mine the network importance of moving objects in this trajectory network.

At any point in time t, we can consider *a proximity network*, defined as a graph with nodes representing objects and edges representing contacts that are currently active. The proximity network satisfies particular distance requirements and its topology depends on the proximity threshold employed. Given a proximity threshold, the trajectories of the moving objects continuously form new contacts, while other contacts are dissolved, leading to a collection of proximity networks. These networks can be understood as a dynamic network the topology of which is changing over time, *a temporal network*. We refer to this temporal network defined by the trajectories of moving objects over time as *a trajectory network*.

In this paper, we are interested in *mining the importance of moving objects* (represented as nodes) in a trajectory network.

The concept of node importance has been rigorously studied in the case of *static graphs*, due to its numerous applications. These include measuring the influence of individuals in a social network, understanding the role of infrastructure nodes in transportation networks, urban networks, the Internet, or assimilating the role of a given node in spreading a contagious disease, to name a few.

Most of the metrics employed to characterize the importance of a node in a static graph depend on the number of direct connections that this node has to other nodes in the graph (i.e., the node degree). In the case of a trajectory network, these connections are evolving over time, therefore the definition of node importance needs to be redefined. For example, the number of unique connections or average number of connections over time might be of interest. In addition, while existence (or not) of a node's connections remains important, the temporal dimension adds more quantities of interest that need to be examined, including metrics of *frequency* and *duration* of these connections. These metrics can be equally important in understanding a node's role in the network and being able to evaluate them allows for an overall more comprehensive analysis. Other metrics of importance include the level of connectedness of a node to other nodes over time, as well as, the membership of a node to specific network motifs, e.g., network triangles. Fig. 1 presents a motivating example of the nature of the problem. We further discuss and formally define these metrics in Section II.

While the temporal dimension adds richness to the analysis, it also adds to the complexity of the graph representation and demands for efficient methods for evaluating the metrics of interest. The naive approach to the problem requires to evaluate the various node importance metrics over a number of *proximity graphs* (assuming observation time is discretized), using traditional static graph algorithms and then aggregating quantities in a meaningful way. This is in addition to applying expensive trajectory similarity methods over multiple points to construct the proximity network at each point in time. There are also streaming versions of algorithms that focus on efficient computation of single network metrics over time. In contrast to these approaches, we devise a novel method that is able to simultaneously evaluate a number of network metrics of interest for all moving objects (i.e., all trajectories), over time. Our proposed method is based on two phases. First, it efficiently computes and represents contacts of moving objects over a period of time as a set of time intervals. Then, a one-pass algorithm is used to evaluate the metrics of interest, all at once, by efficiently processing the sets of time intervals of contacts. In the cases we present, moving objects have either constant velocities or follow random trajectories while other parameters that can dramatically affect the results and the algorithms' performance are considered, such as the proximity threshold τ. Furthermore, our method is utilized to evaluate the membership of an object to interesting network motifs, allowing for a more comprehensive clustering analysis of objects over time. In summary, the major contributions of this work include:

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
$[0, T]$	Observation time interval
$\mathscr{N}$	A set of moving objects
$\mathbf{P}_i$	Trajectory of an object i
(x, y, t)	Coordinates of an object at time t
τ	Proximity threshold
$d_{u,v}$	Spatial distance of u and v
$c_{u,v}$	Contact of u and v
$e_{u,v}$	Event about a contact between u and v
Δt	Duration of an event
$G_{[0,T]}(V, E)$	Trajectory network of V nodes and E edges
V_t	Set of nodes at time t
E_t	Set of edges at time t
$G_t(V_t, E_t)$	Proximity network at time t

- a novel framework for network-based trajectory analysis.
- a novel one-pass algorithm, for fast and accurate mining of node importance in trajectory networks.
- a thorough evaluation of node importance methods on large-scale synthetic data, for a range of conditions.
- making *source code* and *data* publicly available to encourage reproducibility of results.

The remainder of this paper is organized as follows: Section II introduces notation and preliminaries related to the problem. Section III formally defines the problem of interest and its variations. Our methods and overall framework are presented in Section IV. Section V presents the details of our algorithms. After reviewing the related work in Section VII, we conclude in Section VIII.

II. Definitions and Preliminaries

Consider a set of objects $\mathscr{N} = \{1, 2, \ldots, N\}$ moving in the Euclidean plane $\mathbb{R}^2$ for a finite *observation time interval* $[0, T]$, forming trajectories $\mathbf{P}_i, i \in \{1, 2, \ldots, N\}$. As the objects are continuously moving, they can at times encounter each other, forming *contacts*.

Definition 1: (Contact) A *contact* c between two moving objects $u, v \in \mathscr{N}$ occurs when the physical proximity (spatial distance) $d_{u,v}$ of the two objects is smaller than or equal to a threshold τ ($d_{u,v} \leq \tau$). A contact is represented as a pair of nodes $c_{u,v} = (u, v)$.

Several approaches can be used to estimate the spatial distance of two points in Euclidean plane. In this paper, we employ its simplest form, the Euclidean distance, given by:

$$d_{u,v} = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$$

where (x_u, y_u) and (x_v, y_v) are the spatial coordinates of objects u and v at a time t, where $0 \leq t \leq T$ respectively.

As mentioned earlier, a contact is considered *active* for as much as the spatial distance between the two objects u and v remains consistently smaller than a proximity threshold τ. To better describe this concept, we introduce the concept of an *event*.

Definition 2: (Event) An *event* e occurs when two moving objects $u, v \in \mathscr{N}$ form a contact $c_{u,v}$ and is represented by a

pair $e_{u,v} = (c_{u,v}, [t_s, t_e])$, where $c_{u,v} = (u, v)$ and $[t_s, t_e]$ is a time interval between the time at which the contact became active t_s and the time that the contact dissolved t_e. We also refer to the times t_s and t_e as endpoints of an event, the one representing the starting point and the other the ending point of the event e. Endpoints are critical for our proposed methods. For simplicity, we sometimes represent an event as a quadruple $e_{u,v} = (u, v, t_s, t_e)$. An event has also a duration $\Delta t = t_e - t_s$.

Note that, in our setting, we do not preclude the case that two objects contact each other multiple times over the observation time interval $[0, T]$. In this case, a contact between two moving objects u and v is represented by a sequence of events $E_{u,v} = \{e^1_{u,v}, \ldots, e^n_{u,v}\}$ or $E_{u,v} = \{(c_{u,v}, [t^1_s, t^1_e]), \ldots, (c_{u,v}, [t^n_s, t^n_e])\}$, and the respective durations of the events as a set $E^{\Delta t}_{u,v} = \{\Delta t^1, \ldots, \Delta t^n\}$.

In this paper, we employ a universal proximity threshold τ, so the contacts will always be *reciprocal*, meaning that $(c_{u,v}, [t_s, t_e])$ is equivalent to $(c_{v,u}, [t_s, t_e])$. While we can assume that the reciprocity property is valid in many applications (e.g., vehicle-to-vehicle proximity, human-to-human proximity, to name a few), there are interesting cases and applications where the reciprocity property might not always be satisfied. For example, if proximity is defined as the ability of a node to perceive another object, then we can assume that nodes u and v might have a different degree of that skill, and therefore $(c_{u,v}, [t_s, t_e])$ is not necessarily equal to the $(c_{v,u}, [t_s, t_e])$. These cases are out of the scope of this work.

A. Trajectory Networks

Monitoring the physical proximity of moving objects, can be represented as a *trajectory network*. Formally, a trajectory network $G_{[0,T]}$ defined in an *observation time interval* $[0, T]$ consists of a set of vertices $V_{[0,T]}$ and a set of edges $E_{[0,T]}$. It is easy to see that $V_{[0,T]}$ represents all moving objects $\mathcal{N}$ and $E_{[0,T]}$ represents all the *events* that occurred in $[0, T]$.

A trajectory network is inherently dynamic and can also be thought of as a *temporal network*, also referred to as a *time-varying network*. Most characterizations of temporal networks discretize time by converting temporal information into a sequence of n network "snapshots". We use w to denote the time duration of each snapshot (time window size), where $w = T/n$, expressed in some time unit (e.g., seconds, minutes, hours, etc.). For simplicity we assume that $w = 1$. In other words, a temporal network can be represented as a series of static graphs G_1, G_2, ..., G_n. The notation $G_t(V_t, E_t)$, $\forall t \in \{1, 2, \ldots, n\}$ represents the temporal network snapshot at time t, where V_t, E_t are the sets of vertices and edges at time t, respectively. It is easy to see that $G_t(V_t, E_t)$ represents a proximity network at time t, where V_t represents moving objects and E_t represents all active contacts at time t.

Edge Stream Representation of a Trajectory Network: A trajectory network $G_{[0,T]}$ can be represented as an *edge stream* —a sequence of all events $e \in E_{[0,T]}$ ordered by their starting time t_s. For example, if $E_{[0,T]}$ has the following events: $\{(u_1, u_5, \underline{3}, 7)$, $(u_2, u_7, \underline{2}, 4)$, $(u_1, u_2, \underline{5}, 7)\}$, then the edge stream appears as follows: $\{(u_2, u_7, \underline{2}, 4)$, $(u_1, u_5, \underline{3}, 7)$, $(u_1, u_2, \underline{5}, 7)\}$. If two events start at the same time, their ordering is considered arbitrary. Our algorithms assume that the edges are given in chronological order; if not, they can be sorted in $O(mlogm)$ time, where $m = |E_{[0,T]}|$. The *edge stream* is a natural way to represent a trajectory network, e.g. representing walking individuals over time, vehicles moving in city over time, and more. The edge stream of G can be modeled as a set of n continuous line segments (i.e, event time intervals) with freely defined starting and ending points placed along an horizontal axis representing time (see Fig. 2).

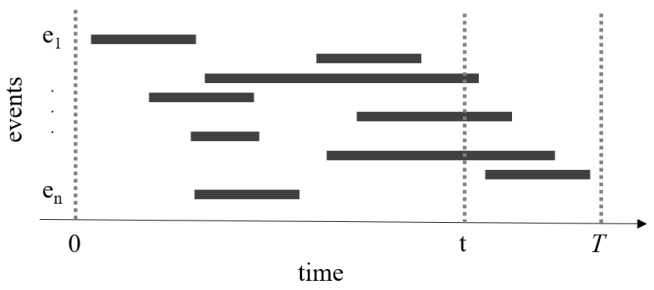


Fig. 2. A set of continuous line segments representing time intervals can be used to model the events that occur in a trajectory network.

B. Node Importance in Trajectory Networks

In this paragraph, we define metrics that relate to the temporal importance of nodes. Note that we abstain from the term *node centrality* to refer to node importance that is common in static network analysis. This is because measures of node centrality in the traditional setting of a static network are commonly based on *shortest paths* (e.g., betweenness centrality [6], [7]), but shortest paths in temporal networks take a different character [8]. For example, in [9], the authors define *minimum temporal paths* to capture the different characterizations of time-constraint shortest paths including cases of earliest-arrival paths, latest-departure paths, or fastest paths. It is possible to evaluate a notion of temporal betweenness [10], but in our setting, we focus on more versatile notions of importance that are critical in the context of trajectories and network-based trajectory analysis. That includes metrics that relate to *the temporal node degree*, *duration* of contacts, *node connectedness*, and *node triangle membership*, as described below. In addition, we describe global metrics that offer insights about the state of the observed system of trajectories, over the observation time period $[0, T]$, including descriptive analysis of the number of events per time and space and network community profiling analysis.

Definition 3: (Trajectory Node Degree) We generalize the concept of a node importance to that of *node profiling* in trajectory networks. For a moving object $u \in \mathcal{N}$, we assume that the object might appear and disappear during the n time units of the observation time interval $[0, T]$. We represent the γ sequences of the continuous periods of presence as: $\Gamma_u = \{[t^1_a, t^1_\omega], \ldots, [t^\gamma_a, t^\gamma_\omega]\}$. Each appearance i of an object

u spans $|\Gamma_u^i| = (t_\omega^i - t_a^i + 1)$ time units, and the total number of observation time units T_u of an object u will be $T_u = \sum_{i=1}^{\gamma} |\Gamma_u^i|$, where $T_u \leq n$. Then, we define the following metrics:

- C_u: a set of all the contacts of u during the observation time interval $[0, T]$.
- deg_u^{Max}: the maximum number of concurrent contacts at some time t, $0 \leq t \leq T$.
- deg_u^{Min}: the minimum number of concurrent contacts at some time t, $0 \leq t \leq T$.
- $deg_u^{Avg} = \frac{\sum_1^{|\Gamma_u|} \sum_{i=1}^{|\Gamma_u^i|} deg_u^t}{T_u}$: the normalized mean temporal node degree, where deg_u^t is the degree of u at time t and $t \in [t_a^i, t_\omega^i]$, $i = \{1, \ldots, \gamma\}$.
- $D_{deg_u}(k)$: the distribution that represents the fraction of the time $[0, T_u]$ that u has node degree k.

In a static undirected graph, a node v is *reachable* from a node u if there is a sequence of adjacent nodes (i.e., *a path*) starting at u and ending at v. In addition, a *connected component* is a subgraph of an undirected graph in which any two nodes are reachable to each other. Node reachability and connectedness are important since they allow to characterize the topology of a network and to investigate the dynamics of processes occurring in it. In the case of a trajectory network $G_{[0,T]}$, node adjacency is a function of time t and a proximity threshold τ, therefore the concepts of reachability and connectedness need to be redefined.

Definition 4: (Trajectory Node Connectedness) Two nodes u and v are *reachable* at time t if there is a sequence of adjacent nodes connecting them in the proximity network at time t. Similarly, a *connected component* cc is a subgraph of the proximity network at time t in which any two nodes are reachable to each other. We also define as *node connectedness* cc_u^t of u the connected component containing u at time t. Then, we can define the following metrics:

- CC_u: a set of all the connected components that contained u during the observation time interval [0,T].
- $D_{CC_u}(k)$: a distribution of the size of connected components that represents the fraction of the time $[0, T_u]$ that u is a member of components of size k.

In a static undirected network, the clustering coefficient of a node u is a fundamental measure that quantifies how close its neighbours are to being a clique. Its computation can be reduced to counting the number of *triangles* incident on the particular node u in the network, where a triangle is a set of three nodes u, v, w such that (u, v), (v, w), (u, w) are edges in the graph. In the case of a trajectory network $G_{[0,T]}$, the number of triangles incident on a particular node u is a function of time t and a proximity threshold τ, therefore the concept of a triangle and membership in a triangle needs to be redefined. Duration of membership to each triangle is also imported.

Definition 5: (Trajectory Node Triangle Membership) A node u is a member of a triangle $\{u, v, w\}$ at time t if there are nodes $u, v, w \in V_t$, such that (u, v), (v, w), $(u, w) \in E_t$ in the proximity network G_t. Then, we can define the following metrics:

- $\lambda_u^{G[0,T]}$: the number of triangles during the observation time interval $[0, T]$ that v is a member of.
- $D_{\lambda_u}(k)$: a distribution that represents the fraction of the time $[0, T]$ that u is a member of k triangles.

It is easy to see that a number of *global trajectory network analytics* are possible in a post-processing phase. For instance, it is possible to perform an enumeration of all the connected components, along with the time duration of each, during the observation time interval $[0, T]$ leading to a complete *network community profiling* [11] analysis for the trajectory network. We skip formal coverage of these metrics due to space limitations, but provide example analysis in the experimental evaluation section.

III. The Problem

In this paper, we are interested in mining the network importance of moving objects in trajectory networks. In the previous section, we have explained how the semantics of network importance had to be redefined to consider the spatio-temporal notion of node degree, node connectedness and node membership in triangles. We collectively refer to the problem of interest as *the Moving Object Network Profiling problem*, or simply *MONetPro*, and we formally define it as follows:

Problem 1: (MONetPro) Given the trajectories $\mathbf{P}_i$, $i \in [1, \ldots, N]$ of $\mathcal{N}$ moving objects, an observation time interval $[0, T]$ and a proximity threshold τ that defines a *non-negligible contact* between two objects, compute the metrics that define:

i. the trajectory node degree of each object.
ii. the trajectory node connectedness of each object.
iii. the trajectory node triangle membership of each object.

IV. Methodology

We first present methods for constructing a trajectory network. Then, we present methods that given an a trajectory network, address the subproblems of Problem 1.

A. Construction of the Trajectory Network

Given a set of trajectories of moving objects, the first task is to construct the trajectory network. The result of this process is an edge stream representation of a trajectory network — a sequence of all events $e \in E_{[0,T]}$. Typically, the construction of the trajectory network requires considerable time. This time depends on the type of motion of the objects that is expected/allowed. Below we cover the case of *random trajectories* and the case of *trajectories of constant velocities*. Both have interesting real-world applications.

Random Trajectories: In the general case, a trajectory is a random walk in the Euclidean space. In that case, in order to construct the trajectory network $G_{[0,T]}(N, E)$ (without assuming any advanced approximation method) we need to compute the distances between all pairs of objects $(u, v) \in V_t \times V_t$ that are present at time t. If the distance is less than or equal to the proximity threshold τ, then an edge is added to the trajectory

network that connects the two nodes at that time. The process is continued for subsequent times, eventually finding all events $e \in E_{[0,T]}$. The computation cost of this process for the entire observation time [0, T] is $O(T \cdot |V_t|^2)$.

Trajectories of Constant Velocity: In many applications, we can assume that objects are moving with constant velocity (i.e., constant speed and direction), forming trajectories that can be represented as straight lines in the Euclidean space. In that special case, the position of the objects as a function of time can be described using a linear equation. This case is interesting because we can resort to an algebraic way of determining *whether* and *when* an edge between two moving objects exists. In practice, we need to determine the *switching times* when the distance of each pair of objects is smaller or larger than the proximity threshold τ, without the need to compute their distance again and again for each time unit. Formally,the position of an object i moving with constant velocity, as a function of time $t \in [0, T]$ can be described by the following two equations:

$$p_{x,i}(t) = s_{x,i} \cdot t + x_{0,i}$$
$$p_{y,i}(t) = s_{y,i} \cdot t + y_{0,i}$$

where $s_{x,i}$ is the speed of object i and $x_{0,i}$ is its initial position in the x axis. Similarly, $s_{y,i}$ is the speed of object i and $y_{0,i}$ is its initial position in the y axis. Then, we can express the distance between two points a and b as a function of time, as:

$$d_{a,b}(t) = \sqrt{(p_{x,a}(t) - p_{x,b}(t))^2 + (p_{y,a}(t) - p_{y,b}(t))^2}$$

where $d_{a,b}(t)$, is the distance of points a and b over time. Now, the solutions (if any) of this quadratic polynomial for $d_{a,b}(t) = \tau$ are the times when a contact is formed or dissolved. This process requires a single distance calculation between any pair of objects $(u, v) \in V \times V$, so the required time is $O(|V|^2)$.

B. Methods for Moving Object Network Profiling

Given a representation of all events over time that define t proximity networks G_t, $t \in [0, T]$, we need to compute the metrics of interest that define the subproblems of Problem 1. Towards this end, we present three approaches: a *naive* approach of applying standard graph algorithms on every proximity network G_t, a *streaming approach* of computing metrics over a stream of edges, and our proposed method that its key idea is based on applying a Sweep Line Over Trajectories (*SLOT*). A sweep line algorithm [12] is an algorithmic paradigm that uses a conceptual sweep line to solve various problems in Euclidean space — it is one of the key techniques in computational geometry.

Naive Approach: The naive way to address the problem is to first construct a set of all the proximity networks $G_t(V_t, E_t)$ for each time unit $t \in [0, T]$. Then, visit each G_t independently and compute the metrics of interest for each node $u \in V_t$ by applying standard graph algorithms on static graphs. Once all networks have been examined, a post-processing step is required that would collect and aggregate the independent results in order to compose the final results. The post-processing phase could be dropped if the G_t networks are examined in a temporal order. In that case, it is possible to update the metrics of interest on-the-fly (progressively) as they are computed in subsequent proximity networks. But, the naive approach has serious drawbacks. First, it will be very inefficient, because each proximity network still needs to be constructed for every time t. Then, the standard algorithms need to be run for T time units, so computations would grow linearly to the size of the observation period $[0, T]$. Given a trajectory network $G_{[0,T]=(V,E)}$, the worst-case computational complexity of the different subproblems is as follows:

- Trajectory Node Degree: $O(T \cdot (|V_t| + |E_t|))$. Every vertex and every edge of each proximity network G_t need to be explored in the worst case, for $t \in [0, T]$.
- Trajectory Node Connectedness: $O(T \cdot (|V_t| + |E_t|))$. The connected components of a proximity network G_t can be found by applying a breadth-first or depth-first search algorithm, which are known to have a computational complexity of $O(|V_t| + |E_t|)$ in worst case [13]. They need to be applied for every $t \in [0, T]$.
- Trajectory Node Triangle Membership: $O(T \cdot (|V_t|^3))$. The trivial approach of counting the number of triangles in a proximity network G_t is to check for every triple $(u, v, z) \in \binom{|V_t|}{3}$ if nodes *u*, *v*, *z* form a triangle. This procedure has a worst-case complexity of $O(|V_t|^3)$. Faster algorithms are also known for finding and counting triangles that rely on fast matrix product and have a computational complexity of $O(|V_t|^\omega)$, where $\omega < 2.376$ [14], [15]. They need to be applied for every $t \in [0, T]$.

Streaming Approach: Since a trajectory network can be represented as an edge stream, an alternative approach to evaluating the metrics of interest is to consider streaming versions of the graph algorithms. The main idea is that at each time $t \in [0, T]$ some of the metrics of interest are computed and this information is carried over to subsequent time units. As a result, unnecessary computations are dropped. The main drawback of the streaming approach is that computations still have to take place at every t. Streaming algorithms would have a similar worst case computational complexity to static algorithms examined in the naive approach. It is easy to see that if one considers the case where all vertices and edges are becoming available in a single time t. As we show in Section VI, in practice, the streaming algorithm would always outperform the naive approach, since it only needs to account for the updates that occur among subsequent times, but its cost would still be dominated by the requirement to run for each $t \in [0, T]$. In Section VI we only experiment with streaming versions for computing the trajectory node degree related metrics, but skip streaming algorithms for more advanced metrics that are not readily available. These are adequate to demonstrate the relative performance of streaming versions to the naive method and our proposed methods.

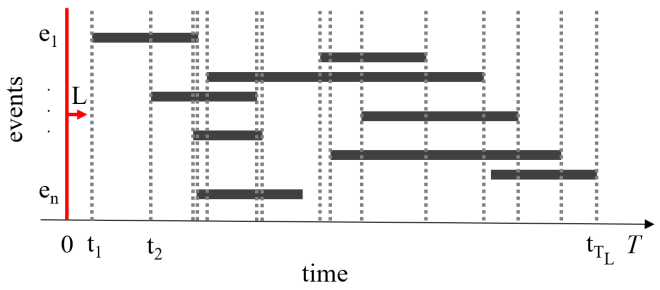


Fig. 3. Illustrative example of *SLOT*, our proposed method that employs a sweep line approach to determine when intersection of events occur. As a result, in the general case, only a small set of time units need to examined

Sweep Line Over Trajectories (SLOT): We can further improve the performance of the streaming algorithms by avoiding a large number of unnecessary computations. The key idea of our proposed method is that even if an event $e_{u,v} = (u, v, t_s, t_e)$ has a duration of $\Delta t = t_e - t_s$ time units, the metrics of interest for each node need only be updated at the **endpoints** t_s and t_e of each event $e_{u,v}$. Recall that the endpoints define when an edge in the network (i.e., a contact) is created or dissolved. When the time intervals (i.e., durations) of multiple events that involve the same node u are overlapping (it is easy to imagine such cases in the example of Fig. 2), then we need to consider these events simultaneously and inform the correct update of the related metrics.

Processing events only at the endpoints (instead of all the time units) has the premise of improving the computational performance of the method by orders of time. However, there is a challenge. In many of our metrics, the duration of the overlapping time of events is increasingly important. In the cases of the *streaming approach* and *the naive approach*, where all units are examined, we had to simply increment the duration values by one (1) time unit, when needed. Now that endpoints are examined in arbitrary times, computing the duration of overlapping times (without examining all time units) is challenging.

To address this problem, we adopt a *sweep line approach*. A sweep line [12] is an algorithmic paradigm that uses a conceptual sweep line, say L, to solve various problems in Euclidean space. Given a set of line segments, similar to the ones in Fig. 2, the line moves from left to right across the plane, intersecting the input line segments in sequence as it moves. In such a case, L will end up intersecting the input line segments (i.e., events) in a number of points (i.e., time units $T_L \leq T$) defined by the finite set of line segment endpoints (i.e., the event endpoints). These are the only points where the sweep line L is intersecting with the line segments, and the process effectively determines the only time units that need to be examined in our problem. Specifically, each of the T_L time units is associated with an event starting time t_s or an event ending time t_e (see Fig. 3). Therefore, our proposed method, *SLOT*, is a one-pass algorithm that employs a variant of the sweep line algorithm and runs in time linear to T_L.

TABLE II
SUMMARY OF TIME COMPLEXITIES

	Naive	Streaming	SLOT
i	$O(T \cdot (\|V_t\| + \|E_t\|))$	$O(T \cdot (\|V_t\| + \|E_t\|))$	$O(\|E\|)$
ii	$O(T \cdot (\|V_t\| + \|E_t\|))$	$O(T \cdot (\|V_t\| + \|E_t\|))$	$O(\|E\|)$
iii	$O(T \cdot (\|V_t\|^3))$	$O(T \cdot (\|V_t\|^3))$	$O(\|E\|)$

Specifically, when any of the T_L time units is examined, a number of quantities are computed that relate to metrics of the subproblems of Problem 1. As a result, the computation cost of our method and its variations will now be a factor of T_L, instead of T. It's important to note that at each time unit examined, only incremental updates of already maintained quantities need to be performed, relevant to the metrics of interest. Given a trajectory network $G_{[0,T]=(V,E)}$, the number of events is the same as the number of edges E, thus the number of event endpoints will be $T_L = 2|E|$. It is sensible to assume that the maximum node degree of our network is much less than the total number of edges ($max(deg_u) \ll |E|$). Therefore, the worst-case computational complexity for all three trajectory node metrics is $O(|E|)$.

Our algorithms assume that the edges are given in chronological order; if not, they can be sorted in $O(|E| \cdot log|E|)$ time.

Table II provides a summary of all the time complexities of different methods for each subproblem of Problem 1. We provide implementation details of the algorithms that compute the various metrics in Section V and demonstrate the efficiency of the method in Section VI.

V. ALGORITHMS

The SLOT algorithm receives a set of all endpoints t_s and t_e of all events $e_{u,v}$ and computes the relevant metrics for all nodes $u \in V$, in a single pass. The algorithm has two parts. In the first part, all variables are initialized using `InitializeMetrics`. In the second part, event endpoints are iterative processed to compute metrics of interest. For each metric, a separate metric procedure is invoked. Details of the metric procedures are provided in the next paragraphs. There is also an auxiliary procedure, `StoreNodeMetric`, which is used by all metric procedures of the algorithm. This procedure computes the elapsed time between the last and current endpoint for a particular node and incrementally updates metric and duration values.

Trajectory Node Degree: For the computation of the trajectory node degree, we provide the `CalculateDegree` procedure. As endpoints are processed, we monitor the current node degrees and increment or decrement them according to whether an edge attached to a node is added or removed.

Trajectory Node Connectedness:

For the computation of the trajectory node connectedness, we provide the `CalculateConnectedNess` procedure. As endpoints are processed, we monitor the currently available components in the network, information about the component

Algorithm 1: Main SLOT algorithm that scans over the sequence of edge events and at the endpoints of each event, calculates the relevant metrics based on user-selected flags.

Input: Set V of nodes, Set $Ends$ of all event endpoints t_s, t_e of all events $e_{u,v} = (u, v, t_s, t_e)$

Output: Set V of nodes, each with a set representing the node's distributions of metric values over time $(u, D_{deg_u}, D_{CC_u}, D_{\lambda_u})$

```
if not sorted(Ends) then
    Sort(Ends)
InitializeMetrics(V)
for endpoint in Ends do
    CalculateDegree(V, endpoint)
    CalculateTriangles(V, endpoint)
    CalculateConnectedness(V, endpoint)
```

Procedure InitializeMetrics(V)

```
components ← []
for node in V do
    node.last_time ← 0
    node.degree.value ← 0
    node.degree.history ← []
    node.neighbors ← []
    node.triangles.value ← 0
    node.triangles.history ← []
    components.append(node)
    node.component ← components[node]
    node.connectedness.value ← 0
    node.connectedness.history ← []
```

Procedure StoreNodeMetric(V, endpoint, node, metric)

```
new_duration ← endpoint.time - node.last_time
if new_duration> 0 then
    if node.metric.value not in node.metric.history then
        append(node.metric.value) to node.metric.history
        node.metric.value.duration ← 0
    node.metric.value.duration + = new_duration
    node.last_time ← endpoint.time
```

Procedure CalculateDegree(V, endpoint)

```
StoreNodeMetric(V, endpoint, endpoint.node_u, degree)
StoreNodeMetric(V, endpoint, endpoint.node_v, degree)
if event.type = start then
    endpoint.node_u.value + = 1
    endpoint.node_v.value + = 1
else
    endpoint.node_u.value − = 1
    endpoint.node_v.value − = 1
```

Procedure CalculateConnectedness(V, endpoint)

```
node_u←endpoint.node_u, node_v←endpoint.node_v
com_u←node_u.component, com_v←node_v.component
if endpoint.type=start then
    if com_u != com_v then
        new_com← MergeComponents(com_u, com_v)
        new_size = length(new_com)
        for node in new_com do
            node.component← new_com
            StoreNodeMetric(V, endpoint, node,
              connectedness)
            node.connectedness.value = new_size
else
    connected, com_u, com_v = BFS(V,node_u,node_v)
    if not connected then
        size_u = length(com_u)
        size_v = length(com_v)
        for node in com_u do
            node.component← com_u
            StoreNodeMetric(V, endpoint, node,
              connectedness)
            node.connectedness.value = size_u
        for node in com_v do
            node.component← com_v
            StoreNodeMetric(V, endpoint, node,
              connectedness)
            node.connectedness.value = size_v
```

each node participates in, and its size. When a new edge is added, there are two possible outcomes: either the two nodes attached to the new edge are already members of the same component (no need to update), or they are members of two different components that now need to be merged into one. In either case, the values of all nodes participating in any of the affected components are updated. When an edge is removed, we must perform a search to check if this has caused a disconnection, or the component remains connected. We perform a breadth-first search to determine if the two nodes are still connected. If after the removal of the edge, the two nodes are still connected, there is no need to update. Otherwise, the current component needs to be split into two components and the values of all nodes participating in any of the affected components are updated.

Trajectory Node Triangle Membership: For the computation of the trajectory triangle membership, we provide the `CalculateTriangles` procedure. As endpoints are processed, we monitor the current number of triangles each node participates in, and in addition, a list of the currently connected neighbors of each node. When a new edge is added (or removed), we check whether the two newly connected (or

Procedure CalculateTriangles(V, endpoint)

```
node_u←endpoint.node_u, node_v←endpoint.node_v
if endpoint.type = start then
    append(node_v) to node_u.neighbors
    append(node_u) to node_v.neighbors
else
    remove(node_v) from node_u.neighbors
    remove(node_u) from node_v.neighbors
for node_i in endpoint.node_u.neighbors do
    if node_i in endpoint.node_v.neighbors then
        StoreNodeMetric(V, endpoint, node_u, triangles)
        StoreNodeMetric(V, endpoint, node_v, triangles)
        StoreNodeMetric(V, endpoint, node_i, triangles)
        if endpoint.type=start then
            node_u.triangles.value += 1
            node_v.triangles.value += 1
            node_i.triangles.value += 1
        else
            node_u.triangles.value -= 1
            node_v.triangles.value -= 1
            node_i.triangles.value -= 1
```

disconnected) nodes have any common neighbors. If they do, then a new triangle is formed or an existing one no longer exists, respectively. We update the metric values accordingly for each of the three nodes, and we update the lists of active neighbors for the two nodes of the new edge.

VI. Experimental Evaluation

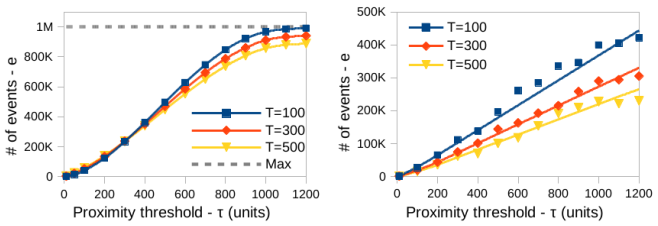
In this Section, we experimentally evaluate the performance of SLOT compared to the static and streaming methods. We also examine the effect of some critical parameters on its computation cost. We aim to answer the following questions:

- **Q1 Effects of the Proximity Threshold.** How does the proximity threshold τ affect the final number of edges and events in the trajectory network?
- **Q2 SLOT Comparative Performance.** How much faster is SLOT compared to the naive and streaming methods?
- **Q3 SLOT Scalability.** How SLOT scales to larger number of objects or events in the trajectory network?

Before presenting the results, we provide details of the computational environment and the data sets employed.

Environment: All experiments are conducted on a PC with 8x Intel(R) Core™ i7-7700 CPU @ 3.60GHz and 64GB memory. Python 3.6 is used and the static graph calculations use the state-of-the-art algorithms for the relevant metrics provided by the *networkx* package. For the algorithm performance evaluation, we measure the average time for 10 executions of each algorithm, and for the study of simulation parameter effects, we take the average results of 10 random seed executions.

Data: In order to evaluate the various conditions and parameters of our algorithms, we had to resort to synthetic data.



(a) Constant trajectories (b) Random trajectories

Fig. 4. Effect of proximity threshold on number of events, $N = 1000$.

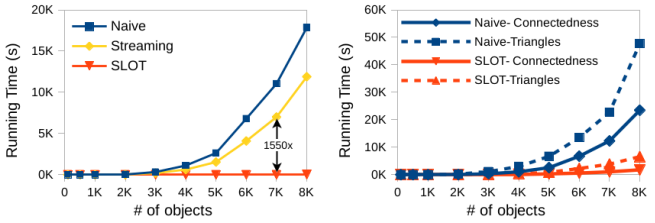
Towards this end, a generator was implemented that allows the simulation of random or constant velocity trajectories over a Euclidean plane. Given the size T of an observation period $[0, T]$ every object is set in motion for T discrete time units forming trajectories. Objects moving outside the boundaries are deleted, while new objects are generated at a steady rate to counteract the loss. The resulting generator has a number of configurable parameters, including *space size*, *min and max speed of an object*, *new object generation rate*, *observation time*. Different combinations of these parameters lead to different trajectory networks. We examine the effect of some of these properties on the results and the algorithm performance. For the majority of the experiments and without loss of generality, we fix the following parameters $space\_size = 1000 \times 1000$, $minimum\_speed = 0$ and $maximum\_speed = \pm 1$. For experimental evaluation purposes, various datasets were created, ranging from 10^2 to 10^5 objects and 10^4 to 10^{10} individual measurements. Additionally, to evaluate the scalability of *SLOT* to larger datasets, we had to resort to a random generator of events (instead of trajectories). This will skip the computations required to construct the trajectory network, a more computationally expensive task.

A. Effects of the Proximity Threshold

We examined the impact of the threshold parameter during the simulation phase on the number of events produced, and therefore on the resulting data size. With fixed values for the rest of the parameters, the results for various threshold values can be seen in Figure 4a for the single-contact case, and Figure 4b for the multi-contact one.

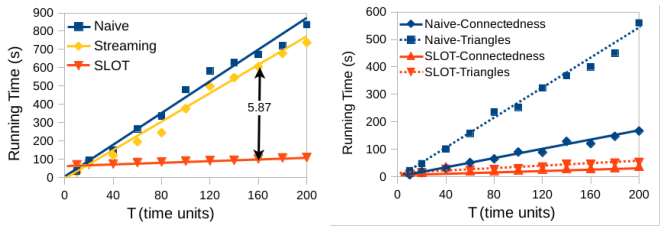
Constant Trajectories: For proximity threshold $\tau < 300$, the number of connections increases quadratically with τ. This happens because the τ defines a circle with radius $r = \tau$ around every object, where any other objects found will be connected to the first. As τ increases, the area of that circle increases relative to the square of τ. Around $\tau = 300$, many objects are limited by the space boundaries and the number of new connections slows, converging to the theoretical maximum of all possible connections $N(N-1)$. Because of objects moving out of bounds, the actual maximum is lower for larger observation times T.

Random Trajectories: In the case of random trajectories, the exact number of events is unpredictable, but it increases in a roughly linear fashion with the proximity threshold.



(a) Node degree (b) Connectedness, Triangles

Fig. 5. Performance of SLOT versus naive, streaming algorithms, $T = 100$.



(a) Node degree (b) Connectedness, Triangles

Fig. 6. Performance of SLOT versus naive, streaming algorithms, $N = 1000$.

B. SLOT Comparative Performance

SLOT is an exact algorithm, so it will always find the correct values of the metrics of interest that it computes. Here, we evaluate the time performance of SLOT against the naive and streaming methods, as a function of the number of objects in the trajectory network. We report results for the trajectory node degree metric. For the rest two metrics (connectedness, triangles), we only compare SLOT to the static method, since streaming versions of algorithms that compute these metrics are not readily available. Fig. 5a presents the results for for trajectory node degree, and Fig. 5b for the rest of the metrics.

SLOT outperforms both the naive and streaming methods by a significant margin, up to $1550\times$ in the case of $7{\cdot}10^3$ objects. This happens because SLOT scales with the number of events $|E|$, which is relative to the number of edges $|E_t|$, but the other algorithms scale with the product of time and number of edges $T{\cdot}|E_t|$, as mentioned in section IV. It can also be seen that the streaming method performs better than the naive. This is to be expected, as the streaming method avoids many unnecessary computations by maintaining information over time. We also experiment with different observation times T, and the results can be seen in Fig. 6. SLOT once more outperforms the naive and streaming algorithms by a large margin, for example $5.87\times$ for $T = 160$. The naive and streaming algorithms behave this way because they scale linearly with time T, while its effect on the calculation cost of SLOT is negligible.

C. SLOT Scalability

Using the synthetic data generated by the trajectory network generator, we examine the performance of the SLOT algorithm itself for different numbers of objects. To obtain consistent results that scale with the number of nodes, the data generated had on average of 20 events per node. This means that the

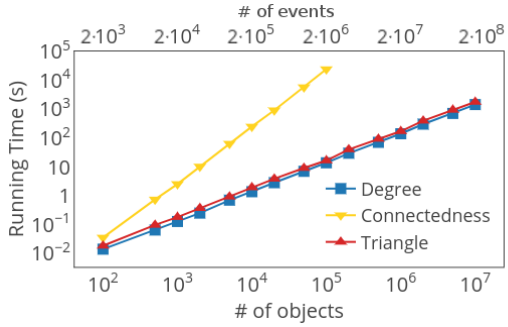


Fig. 7. Performance of SLOT for large scale data, $T = 10000$.

number of events scales linearly with the number of objects, and as a result the execution time of SLOT does the same. This can be seen in Figure 7.

VII. Related Work

Our work is related to *trajectory data mining*, *dynamic* and *temporal networks*. A number of key ideas in these areas have been mentioned throughout this paper, and here we present a more comprehensive view of existing work on these topics.

Trajectory Data Mining: Trajectory problems have been extensively studied in the data mining area. Of particular interest are problems related to similarity in trajectory data [16], [17] and trajectory clustering [18]. While there is research on the behavior and trajectories of moving objects [19], there hasn't been much focus on the interactions between them.

Spatial Networks: The notion of objects distributed in space and interacting with each other has been heavily explored in graph theory. Graph theory concepts such as proximity graphs [20] and geometric intersection graphs [21] are characteristic examples of this. Specifically, several variations of proximity graphs have been developed over time to better fit different problems. For instance, relative neighbor graphs [22] and Gabriel graphs [20] connect nearest neighbors if no other vertexes are nearby, while Delaunay triangulations [23] maximize the minimum angles of all triangles formed. These, however, mostly deal with static data, while our goal was to examine cases of proximity graphs where all objects/nodes are moving and their relationships are evolving over time.

Temporal Networks: There has also been significant research on networks that are evolving over time, or temporal networks. The nature of these systems introduces a number of issues and obstacles, which make it necessary to adapt for the problem basic graph theory concepts and algorithms such as shortest paths [9], motifs [24] and other metrics [25]. Furthermore, with the addition of temporal information, several concepts can be extended to take advantage of the additional data. Examples of this are the temporal node centrality [10] and the network reachability [26] metrics. As part of this project we developed techniques that provide accurate values for these metrics in a fast and accurate way for all nodes, with detailed results over time, then apply these for the analysis of trajectory

temporal networks. In addition to node degree and reachability or connectedness metrics, we also examine the participation of every node in triangles. Earlier research on this topic, including *network temporal motifs*, focused mostly on faster approximations for streaming graphs [27]–[29]. An accurate triangle counting algorithm without approximation exists [30], but merely *counts* all triangles rather than *enumerating* them. Our approach, therefore, allows for more rich analysis of extracted information. Moreover, our algorithm reports the duration of each triangle and does not preclude the case where three nodes may form a triangle several times, something that is typically ignored by existing algorithms.

VIII. Conclusions

In this paper, we have represented trajectories of moving objects as a *trajectory network*. Based on this representation, we have introduced metrics of network importance of moving objects, including *trajectory node degree*, *trajectory node connectedness* and *trajectory node triangle membership*. These metrics can be used to better understand the behavior of a moving object through its interactions with the environment (other objects), over time. They can also be used to reveal interesting network dynamics of moving objects, not easily observable before. In order to evaluate the metrics of interest, we proposed *SLOT*, a fast and accurate algorithm for mining node importance in trajectory networks. Extensive experiments were performed to demonstrate the effectiveness of our algorithm, in a wide range of conditions. Furthermore, our algorithm outperformed naive and sensible streaming approaches to address the problem, by many orders of time. We also demonstrated that our method can scale to very large amounts of trajectories. This work is a substantial first step in understanding network dynamics of moving objects. We are confident that, as large amounts of trajectory data become available, these methods will prove useful and find interesting application in a number of real-world problems and solutions.

Reproducibility: The source code and some execution examples are publicly available to encourage reproducibility of results. They can be accessed at the following website: https://github.com/tipech/trajectory-networks.

Acknowledgments: This research has been supported by a Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant (#RGPIN-2017-05680).

References

[1] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang, "A review of moving object trajectory clustering algorithms," *Artificial Intelligence Review*, vol. 47, no. 1, pp. 123–144, 2017.

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

[3] D. Guo, S. Liu, and H. Jin, "A graph-based approach to vehicle trajectory analysis," *Journal of Location Based Services*, vol. 4, no. 3-4, pp. 183–199, 2010.

[4] K. Siła-Nowicka, J. Vandrol, T. Oshan, J. A. Long, U. Demšar, and A. S. Fotheringham, "Analysis of human mobility patterns from gps trajectories and contextual information," *International Journal of Geographical Information Science*, vol. 30, no. 5, pp. 881–906, 2016.

[5] F. Zanlungo, T. Ikeda, and T. Kanda, "Potential for the dynamics of pedestrians in a socially interacting group," *Physical Review E*, vol. 89, no. 1, p. 012811, 2014.

[6] M. E. Newman, "A measure of betweenness centrality based on random walks," *Social networks*, vol. 27, no. 1, pp. 39–54, 2005.

[7] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[8] D. Kempe, J. Kleinberg, and A. Kumar, "Connectivity and inference problems for temporal networks," in *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. ACM, 2000, pp. 504–513.

[9] H. Wu, J. Cheng, S. Huang, Y. Ke, Y. Lu, and Y. Xu, "Path problems in temporal graphs," *Proceedings of the VLDB Endowment*, vol. 7, no. 9, pp. 721–732, 2014.

[10] H. Kim and R. Anderson, "Temporal node centrality in complex networks," *Physical Review E*, vol. 85, no. 2, p. 026107, 2012.

[11] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 695–704.

[12] M. I. Shamos and D. Hoey, "Geometric intersection problems," in *17th Annual Symposium on Foundations of Computer Science*. IEEE, 1976, pp. 208–215.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[14] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theoretical Computer Science*, vol. 407, no. 1-3, pp. 458–473, 2008.

[15] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, 1997.

[16] M. V. Kreveld and J. Luo, "The definition and computation of trajectory and subtrajectory similarity," *Proc. of the 15th ACM int. symposium on Advances in geographic information systems - GIS 07*, 2007.

[17] K. Toohey and M. Duckham, "Trajectory similarity measures," *SIGSPATIAL Special*, vol. 7, no. 1, pp. 43–50, 2015.

[18] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering," *Proceedings of the ACM SIGMOD international conference on Management of data - SIGMOD 07*, 2007.

[19] S. Dodge, R. Weibel, and E. Forootan, "Revealing the physics of movement: Comparing the similarity of movement characteristics of different types of moving objects," *Computers, Environment and Urban Systems*, vol. 33, no. 6, pp. 419–434, 2009.

[20] K. R. Gabriel and R. R. Sokal, "A new statistical approach to geographic variation analysis," *Systematic Zoology*, vol. 18, no. 3, p. 259, 1969.

[21] T. Erlebach, K. Jansen, and E. Seidel, "Polynomial-time approximation schemes for geometric intersection graphs," *SIAM Journal on Computing*, vol. 34, no. 6, pp. 1302–1323, 2005.

[22] G. T. Toussaint, "The relative neighbourhood graph of a finite planar set," *Pattern Recognition*, vol. 12, no. 4, pp. 261–268, 1980.

[23] B. Delaunay, "Sur la sphre vide. a la mmoire de georges vorono," *Bulletin de l'Acadmie des Sciences de l'URSS*, no. 6, pp. 793–800, 1934.

[24] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, "Temporal motifs in time-dependent networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, no. 11, p. P11005, 2011.

[25] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, "Graph metrics for temporal networks," in *Temporal networks*. Springer, 2013, pp. 15–40.

[26] P. Holme, "Network reachability of real-world contact sequences," *Physical Review E*, vol. 71, no. 4, p. 046119, 2005.

[27] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler, "Counting triangles in data streams," in *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2006, pp. 253–262.

[28] Z. Bar-Yossef, R. Kumar, and D. Sivakumar, "Reductions in streaming algorithms, with an application to counting triangles in graphs," in *Proc. of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2002, pp. 623–632.

[29] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 837–846.

[30] A. Paranjape, A. R. Benson, and J. Leskovec, "Motifs in temporal networks," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017, pp. 601–610.