Scalable Detection of Crowd Motion Patterns

Stijn Heldens, Nelly Litvak, and Maarten van Steen, *Senior Member, IEEE*

Abstract—Studying the movements of crowds is important for understanding and predicting the behavior of large groups of people. When analyzing crowds, one is often interested in the long-term macro-level motions of the crowd as a whole, as opposed to the micro-level short-term movements of individuals. A high-level representation of these motions is thus desirable. In this work, we present a scalable method for detection of *crowd motion patterns*, i.e., spatial areas describing the dominant motions within crowds. For measuring crowd movements, we propose a fast, scalable, and low-cost method based on proximity graphs. For analyzing crowd movements, we utilize a three-stage pipeline: (1) represents the behavior of each person at each moment in time as a low-dimensional data point, (2) cluster these data points based on spatial relations, and (3) concatenate these clusters based on temporal relations. Experiments on synthetic datasets reveals our method can handle various scenarios including curved lanes and diverging flows. Evaluation on real-world datasets shows our method is able to extract useful motion patterns which could not be properly detected by existing methods. Overall, we see our work as an initial step towards rich pattern recognition.

Index Terms—Crowd analytics, pedestrian dynamics, spatio-temporal clustering, trajectory clustering, proximity graph

1 Introduction

Crowds are a common sight these days at busy public locations such as airport terminals, soccer stadiums, or city centers. Various studies (see survey by Castellano et al. [1]) have shown that, even though the behavior of individuals is often erratic and unpredictable, the behavior of a crowd as a whole is usually highly organized and certain spatio-temporal patterns appear at a macro scale which are not visible at the micro scale.

To analyze the movements of crowds, it is thus desirable to aggregate the individual motions into a compact representation that exposes the high-level patterns which emerge from the microscopic movements. One can think of many applications of such a representation, for example, to improve safety of large public events, provide guidelines for urban planners to improve public spaces, or to automate the detection of anomalies.

To further illustrate this concept, consider a busy town square, such as shown in Fig. 1a, and imagine we observe from a birds-eye-view. We discover that most people wander randomly and motions look unstructured at this low level, see Fig. 1b. However, when aggregating these movements, certain *motion patterns* appear and the *collective* movements of the crowd can be described using just a small number of these patterns, see Fig. 1c. While this description does not accurately describe each individual, it does provide an aggregate view of the movements of the entire crowd.

Designing a framework for detecting these crowd motions patterns presents two challenges.

First, there is the problem of how to measure the crowd's movements. The data-collection method should be inexpensive, power-efficient, scalable, and allow holistic analysis of massive indoor/outdoor areas. Commonly used data collection techniques, such as surveillance cameras or GPS receivers, do not meet these requirements.

Second, we are presented with the problem of how to extract motion patterns from the location data. Formalizing a detection algorithm is challenging since the data is noisy and borders between patterns are often not well-defined.

In this work, we present a framework for the detection of motion patterns from real-world crowds. Our method focuses on scalability and resilience, allowing it to scale to thousands of people and be usable in the real world.

For data collection, we follow the ideas by Martella et al. [2] of describing the *texture* of crowds using *proximity graphs*. Proximity mining provides a low-cost and highly scalable method for obtaining movement data. Previous work (Section 6) has shown that proximity graphs can be reliably constructed using low-power sensors. To determine the positions of the nodes over time, we use a fast embedding algorithm to embed the nodes into euclidean space.

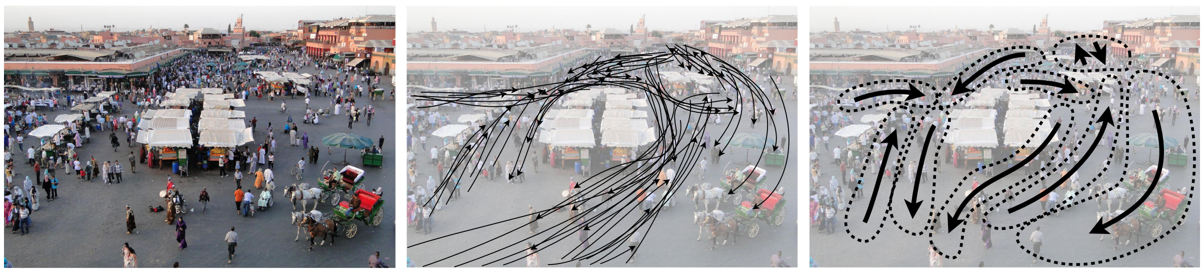
For data processing, our solution relies on two key insights. First, instead of considering entire trajectories, we analyze subtrajectories over small time intervals, resulting in low-dimensional data points (called *tracklets*) describing the local behavior of each person at each moment in time. Second, instead of directly aggregating these low-level tracklets into high-level patterns, we employ a two-stage solution which considers the spatial and temporal relations between tracklets, separately.

We evaluated our method on both synthetic and real-world datasets. For synthetic evaluation, we consider four challenging scenarios: curved lanes, parallel lanes, crossing lanes, and diverging/converging lanes. For real-world evaluation, we show how our method extracts useful patterns from two datasets commonly used in trajectory clustering.

The remainder of this manuscript is structured as follows: Section 2 provides a problem description and a brief overview

• The authors are with the University of Twente, Enschede, NB 7522, Netherlands. E-mail: {s.j.heldens, n.litvak, m.r.vansteen}@utwente.nl.

Manuscript received 11 Mar. 2018; revised 8 Oct. 2018; accepted 14 Oct. 2018. Date of publication 1 Nov. 2018; date of current version 5 Dec. 2019.
(Corresponding author: Stijn Heldens.)
Recommended for acceptance by J. Tang.
Digital Object Identifier no. 10.1109/TKDE.2018.2879079



(a) Overview. (b) Individual paths show motion at micro scale. (c) Motion patterns show motion at macro scale.

Fig. 1. Example of motion patterns at busy city square in Marrakesh, Morocco. Photo by Adam Jones, licensed under CC BY-SA 3.0.

of our method, Section 3 explains our processing pipeline in detail, Section 4 discusses the computational complexity and parameters of our framework, Section 5 presents results, Section 6 contains related work, and Section 7 is dedicated to conclusions and future work.

2 General Overview

In this section, we discuss the two problems of motion pattern analysis: how to measure the crowd's motions and how to aggregate these motions into high-level patterns.

2.1 Data Collection

To enable analysis of crowd motions, we require a data collection method that captures the movements of people in a crowd. While GPS receivers or video cameras could be used for this purpose, we explore another option.

For this work, we follow the ideas by Martella et al. and utilize *proximity graphs* to capture the *texture* of a crowd [3]. A proximity graph is a spatio-temporal graph where nodes represent proximity sensors and edges are proximity detections. Time is discretized into fixed-sized timesteps and two nodes are connected at certain timestep if the corresponding sensors detected each other's presence at that moment in time. In general, we assume two types of sensors: *anchor* sensors, which have a static location, and *mobile* sensors, which are worn by individuals.

Proximity graphs do not store any position data. Detection of motion patterns requires at least some indication of the physical locations of the sensors. These locations need not be highly accurate, since our overall goal is not to position the sensors, but instead, obtain some rough estimate of their motions (i.e., "position over time"). In our framework, we use a fast embedding algorithm to place the nodes into a d-dimensional space. Embedding is repeated at every timestep, each time adapting the locations from the previous timestep using the topology of the next timestep. It is crucial that the embedding is scalable, since it is executed repeatedly and the number of nodes can be large.

Proximity graphs provide a number of advantages over alternative crowd monitoring techniques, such as surveillance cameras or GPS receivers. Proximity sensing is highly scalable, low cost, energy efficient, and requires very little infrastructure to set up (i.e., only the anchors need to be placed beforehand). Surveillance cameras require expensive infrastructure, video processing is computationally expensive, and analysis is limited to the perspective of each camera. GPS receivers [4] are expensive, energy inefficient, and work poorly for indoor/crowded environments, due to radio waves being attenuated by walls and human bodies. Previous research has shown that proximity graphs can be reliably constructed from real-world measurements utilizing smartphones or specialized low-power electronic badges [5].

One of the challenges of proximity mining is that (some subset of) the crowd members needs to be equipped with proximity sensors. For closed-off environments (e.g., festivals, conferences), this can be achieved by distributing electronic "badges" near the entrance. This approach was previously employed for real-world experiments at an IT conference [3] and an art museum [5]. For open environments (e.g., shopping malls, city squares), smartphones could be utilized. There has been research [6] into using bluetooth and Android smartphones for this purpose.

2.2 Motion Pattern Detection

Embedding of the proximity graph provides an estimation of the position of each node at each discrete timestep. Motion patterns can be detected by considering the complete trajectories for the nodes. However, extracting motion patterns directly from trajectories is challenging due to the following issues.

(a) People can transition between different patterns over time. For example, when monitoring a train station, we could track a pedestrian walking through the main entrance, moving up using the escalator, and finally entering the platform. Each of these events corresponds to one motion pattern ("enter station", "use escalator", "move to platform"), but one person contributes to each of these in sequence.

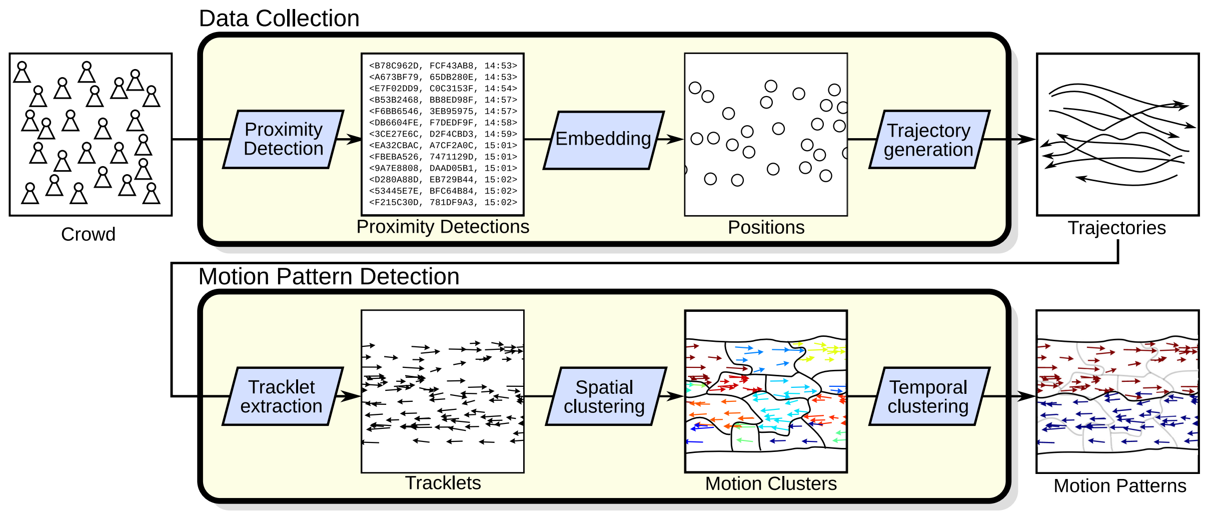
(b) Different individuals contribute to motion patterns at different moments in time. For example, people will use the escalator at different times, thus showing the same behavior in the spatial domain but at different offsets in the temporal domain.

(c) Motion patterns can be any arbitrary shape and they are often elongated. For example, for an escalator, a single cohesive pattern should be detected and not multiple isolated "patches".

(d) For motion patterns, we are interested in the *dominant* motions. Only frequently occurring paths should be detected and rarely used paths should be considered noise and thus discarded. Additionally, more noise is introduced by inaccurate measurements, faulty sensors, different walking speeds, etc. Proper motion pattern detection thus requires resilience against such noise.

To detect motion patterns, instead of directly clustering the trajectories, we propose a three-stage solution:

1) *Tracklet Extraction.* For each node at each timestep, we consider the subtrajectory over a small time window and construct a data point (called *tracklet*) in



```
<B78C962D, FCF43AB8, 14:53>
<A673BF79, 65DB280E, 14:53>
<E7F02DD9, C0C3153F, 14:54>
<B53B2468, BB8ED98F, 14:57>
<F6BB6546, 3EB95975, 14:57>
<DB6604FE, F7DEDF9F, 14:58>
<3CE27E6C, D2F4CBD3, 14:59>
<EA32CBAC, A7CF2A0C, 15:01>
<FBEBA526, 7471129D, 15:01>
<9A7E8808, DAAD05B1, 15:01>
<D280A88D, EB729B44, 15:02>
<53445E7E, BFC64B84, 15:02>
<F215C30D, 781DF9A3, 15:02>
```

Fig. 2. The processing pipeline of our framework. For this particular example, our method detects two opposing movement streams.

a low-dimensional space. A tracklet can be seen as a description of microscopic behavior such as "turned left at the elevator" or "moved up the stairs". This phase targets challenge (a), since long trajectories are split into many tracklets.

2) *Spatial Clustering*. Next, these tracklets are grouped into *motion clusters*, i.e., small "patches" of cohesive behavior. This phase considers only the spatial aspect of the tracklets and ignores the temporal aspect, thus clustering the same behavior at different moments in time and solving issue (b).
3) *Temporal Clustering*. Large motion patterns could get split into multiple smaller motion clusters. To combat this, the final stage of the pipeline concatenates multiple smaller clusters to create larger patterns based on the temporal coherence between clusters. Clusters that are frequently visited in sequence should belong to the same pattern and are combined, resolving issue (c).

Challenge (d) is tackled by careful choice of the algorithms used in each stage.

3 Processing Pipeline

In this section, we describe each phase of our processing pipeline (Fig. 2) in detail.

3.1 Data Collection

3.1.1 *Proximity Detection*

We assume (a subset of) the crowd is equipped with proximity sensors. Additionally, a small number of *anchor* sensors is placed to provide fixed points of reference. Time is discretized into fixed-sized timesteps (e.g., several seconds) and sensors report their detections at every timestep. The exact method for measuring of proximity is out of the scope of this work.

The results of the proximity detection is a dynamic proximity graph $G = (V, E(T))$ where $V = \{v_1, \dots, v_n\}$ are the n sensors, $T = \{1, 2, \dots, t_{\max}\}$ is the set of $t_{\max}$ timesteps, and each undirected edge $(v_i, v_j) \in E(t)$ indicates that nodes v_i and v_j were within proximity of each other at timestep $t \in T$.

3.1.2 *Embedding*

We embed the nodes into a d-dimensional space and estimate the position $p_i(t) \in \mathbb{R}^d$ for each node v_i at every timestep t. We assume either $d = 2$ (for mostly flat environments) or $d = 3$ (for multi-floor buildings). Nodes are initially placed randomly in space and, for each timestep t, an embedding procedure is executed which considers the positions $p_i(t-1)$ together with the edges $E(t)$ and adjusts the node's locations.

It is crucial that this embedding procedure is fast and scalable, since it is repeated at each timestep and should be able to handle massive crowds. Classic graph embedding algorithms, such as force-directed graph drawing [7] and multi-dimensional scaling (MDS) [8], yield high-quality results but are expensive since their run-time is in $\mathcal{O}(n^2)$.

Instead, our embedding method is based on *Stochastic Proximity Embedding* (SPE) [9], which has proven to be usable for large datasets. SPE is based on rounds. During each round, a random pair of nodes (v_i, v_j) is selected and the positions p_i and p_j are adjusted such that their distance $d_{ij} = \|p_i - p_j\|$ more closely matches some "ideal" distance d^*_{ij}. This is achieved by pushing the nodes further apart (if $d_{ij} < d^*_{ij}$) or pulling them closer together (if $d_{ij} > d^*_{ij}$).

We made two modifications to adapt the original SPE algorithm for proximity graphs. First, we defined a heuristic to estimate d^*_{ij} since this information is not provided by the proximity graph. For this work, we define $d^*_{ij} = d_{\text{hop}} h_{ij}$ where d_{hop} is a parameter for the "average single-hop distance" (i.e., mean distance between two nodes in proximity of each) and h_{ij} is the number of hops between the nodes (i.e., length of the shortest path in G at time t) which can be calculated using breadth-first search.

Second, our implementation of SPE considers *only* node pairs (v_i, v_j) which are either 1-hop neighbors ($h_{ij} = 1$) or 2-hop neighbors ($h_{ij} = 2$). This modification was made since it is more important for the embedding to be accurate for nodes which are close (i.e., h_{ij} is small) than far apart (i.e., h_{ij} is large). We experimented with considering *only* 1-hop neighbors, but this showed poor results since the embedding would often "collapse" into itself. We also experimented with 3-hop or even 4-hop neighbors, but this did not improve the embedding quality, while significantly raising the computational effort.

Algorithm 1 shows the pseudo-code for the embedding algorithm. The algorithm performs N rounds. During each round, the algorithm randomly selects pair (v_i, v_j) having $h_{ij} \le 2$, calculates their target distance d^*_{ij}, calculates their current distance d_{ij}, and adjusts the positions of the mobile nodes based on difference between d_{ij} and d^*_{ij}. Parameter λ is the *learning rate*, it decreases by factor ε in each round and aims to avoid oscillation behavior [9].

Algorithm 1. Embedding Algorithm

```
Input: $N$, $d_{\text{hop}}$, $\epsilon$, $p_i(t-1)$ for all $v_i \in V$,
  $S = \{(v_i, v_j) \in V \times V | h_{ij} = 1 \text{or} h_{ij} = 2\}$
Output: $p_i(t)$ for all $v_i$
  $\lambda \leftarrow 1$
  $p_i(t) \leftarrow p_i(t-1)$ for all $v_i \in V$
  loop N times
    $v_i, v_j \leftarrow$ randomly select pair from $S$
    $d^*_{ij} \leftarrow h_{ij} \times d_{\text{hop}}$
    $d_{ij} \leftarrow \|p_i(t) - p_j(t)\|$
    $\Delta \leftarrow \frac{d^*_{ij} - d_{ij}}{d_{ij}}(p_i(t) - p_j(t))$
    if node $v_i$ is non-anchor then
      $p_i(t) \leftarrow p_i(t) + \frac{1}{2}\lambda\Delta$
    end if
    if node $v_j$ is non-anchor then
      $p_j(t) \leftarrow p_j(t) - \frac{1}{2}\lambda\Delta$
    end if
    $\lambda \leftarrow \lambda(1 - \epsilon)$
  end loop
```

3.1.3 Trajectory Generation

The embedding provides an estimate of the location $p_i(t) \in \mathbb{R}^d$ for each node at every timestep t. The trajectory for node v_i is defined as the sequence $(p_i(1), p_i(2), \ldots, p_i(t_{\max}))$.

3.2 Motion Pattern Detection

3.2.1 Tracklet Extraction

To describe the *behavior* of node v_i at timestep t, we consider the subtrajectory $(p_i(t-w), \ldots, p_i(t+w))$, where w is some small predefined integer constant.

Motion clusters are detected by considering the entire set of subtrajectories for all nodes at all timesteps and grouping them into clusters. However, clustering the subtrajectories directly is computationally expensive. Instead, we reduce each subtrajectory into a low-dimensional description called a *tracklet*. This preprocessing step helps to reduce noise and lowers the computational cost of the clustering.

The tracklets should capture at least three characteristics: the average location, the movement direction, and movement velocity. For this work, we define a tracklet $\tau_{i,t}$ for node v_i at time t as the linear approximation in the least squares sense. Least squares fitting comes down to finding the vectors $\hat{\text{p}}_{i,t}$ and $\hat{\text{v}}_{i,t}$ that best fit the points of the subtrajectory for node v_i around time t:

$$p_i(t+k) \approx \hat{\text{p}}_{i,t} + k\hat{\text{v}}_{i,t} \quad \text{for } k \in \{-w, -w+1, \ldots, w-1, w\}.$$

We define this tracklet as $\tau_{i,t} = (\hat{\text{p}}_{i,t}, \hat{\text{v}}_{i,t})$. The vectors $\hat{\text{p}}_{i,t}$ and $\hat{\text{v}}_{i,t}$ can be interpreted as estimations for the location and the movement vector of node v_i at time t.

Fig. 3 demonstrates tracklet extraction for a noisy trajectory. The figure shows how increasing the window size w reduces noise, but also removes finer details and sharp corners. Note that the time windows used for the tracklets are *not* disjoint but overlap. For example, the tracklets at timesteps t and $t+1$ use time windows $[t-w, t+w]$ and $[t-w+1, t+w+1]$, sharing $2w-1$ of the same points.

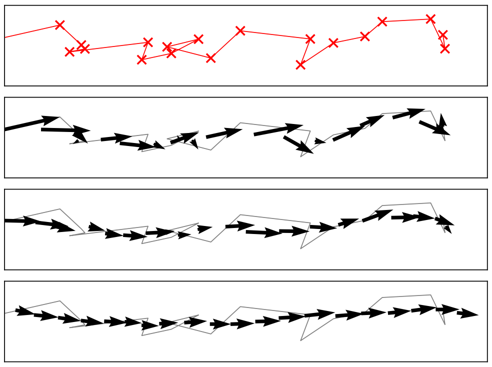


Fig. 3. Example of noisy trajectory (top) and result of tracklets extraction for window size of $w = 1$, $w = 2$, and $w = 3$, respectively. Each arrow visualizes a tracklet $\tau_{i,t}$ at location $\hat{\text{p}}_{i,t}$ having angle/length $\hat{\text{v}}_{i,t}$.

3.2.2 Spatial Clustering

The next step of the processing pipeline is to aggregate similar tracklets into clusters, resulting in small "patches" of cohesive motion. Selecting a clustering algorithm presents the following challenges: (1) the dataset contains noise since human behavior is unpredictable, (2) scalabililty is crucial since the number of tracklets can be large, and (3) there are no clear boundaries between patterns. Common clustering methods are thus unsuitable: k-means [10] does not deal well with noise, hierarchical clustering [11] or mean-shift [12] is too expensive, and SLINK [13] or DBSCAN [14] detect one giant cluster since they are based on transitivity.

Instead, our solution is based on *Quick Shift* [15], a fast nonparametric mode-seeking algorithm which aims to find the local maxima (i.e., modes) of a density function. Each cluster is defined by one of these modes and data points are assigned to each mode using a fast hill-climbing-like procedure. Vedaldi & Soatto [15] coined the name "Quick Shift", although nearly identical methods were proposed by Rodriguez & Laio [16] in 2014 and Koontz et al. [17] in 1976. We borrow notations and conventions from the work by Rodriguez & Laio [16].

Let $\mathcal{T}$ be the set of collected tracklets. We define the *distance* ℓ_{sr} between tracklets τ_s and τ_r as in Eq. (1), taking into account difference in position and velocity. Note that the subscript of τ_s is a tuple indicating a combination of node and timestep, i.e., $s = (i, t)$ for node v_i at timestep t.

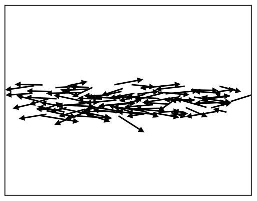
$$\ell_{sr} = \max\left\{\frac{\|\hat{\text{p}}_s - \hat{\text{p}}_r\|}{\alpha}, \frac{\|\hat{\text{v}}_s - \hat{\text{v}}_r\|}{\beta}\right\}. \quad (1)$$

The parameters α and β are used to normalize the position and velocity. We define the local *density* ρ_s for each tracklet as.

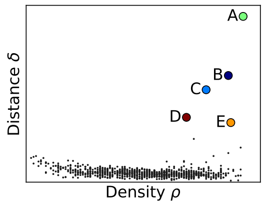
$$\rho_s = \sum_{\tau_r \in \mathcal{T}, \ell_{sr} \leq 1} \|\hat{\text{v}}_r\|. \quad (2)$$

The density function counts similar tracklets (i.e., $\ell_{sr} \leq 1$, or equivalently $\|\hat{\text{p}}_s - \hat{\text{p}}_r\| < \alpha$ and $\|\hat{\text{v}}_s - \hat{\text{v}}_r\| < \beta$), weighting them by $\|\hat{\text{v}}_r\|$. The weighing is needed since tracklets generated by fast-moving nodes are further apart than tracklets for slow-moving nodes, meaning their density (i.e., tracklets per area) needs to be compensate for based on their velocity.

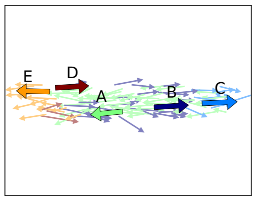
Finally, we define $H(\tau_s)$ as the tracklet of higher density *closest* to tracklet τ_s. Furthermore, let δ_s be the distance ℓ_{sr} between tracklet τ_s and $\tau_r = H(\tau_s)$. By convention, if ρ_s is the global maximum, we define $H(\tau_s) = \perp$ and $\delta_s = \infty$.



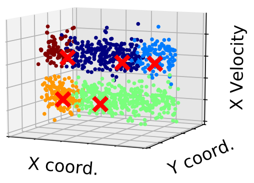
(a) Tracklets.



(b) Density-distance plot.



(c) Cluster centers. Colors show the different clusters.



(d) Tracklets in 3D space. Colors show clusters, marks are centers.

Fig. 4. Example of spatial clustering for narrow hallway.

$$\delta_s = \min_{\tau_r, \rho_s < \rho_r} \ell_{sr}, \quad H(\tau_s) = \arg\min_{\tau_r, \rho_s < \rho_r} \ell_{sr}. \tag{3}$$

As observed by Rodriguez & Laio [16], if point τ_s is a local maximum in the density function then the value of δ_s will be abnormally large since a "jump" is required to go from the local maximum in one region to another region of higher density. Tracklets for which δ_s is greater than some threshold $\delta_{\max}$ are thus defined as *cluster centers* and each is assigned to a unique cluster. Every remaining tracklet τ_s is assigned to the same cluster as tracklet $H(\tau_s)$.

Spatial clustering is thus performed as follows. First, calculate ρ_s for each tracklet τ_s using Eq. (2). Next, calculate δ_s for each tracklet τ_s using Eq. (3). Each tracklet for which $\delta_s > \delta_{\max}$ is marked as a *cluster center* and placed into a unique cluster. Each non-center tracklet τ_s is assigned to the same cluster as $H(\tau_s)$, possibly requiring recursion in order to reach a cluster center. To combat noise, clusters where the center τ_s has low density (i.e., $\rho_s < \rho_{\min}$) are deleted. Some tracklets from $\mathcal{T}$ are thus labeled as noise.

We use an artificial example to motivate our choice for this method and illustrate the underlying intuition. Consider a narrow hallway with pedestrians moving both east-to-west and west-to-east. Applying spatial clustering should ideally reveal two motion patterns for the two directions.

Fig. 4a shows the tracklets for such an area from a top-down view. Fig. 4b plots the densities ρ_s versus the distances δ_s. The plot shows that five points have an abnormally high value for δ_s and are thus centers. Fig. 4c shows the five clusters indicated by different colors with the centers highlighted. This example reveals that the two motion patterns have been split into five clusters: three clusters for eastward motions and two for westward motions.

To understand these clusters, we visualize the tracklets in three dimensions (Fig. 4d) where the Z-axis shows horizontal velocity. The plot reveals two "point clouds" where each cluster centers corresponds to a point inside these clouds having maximal density. Defining clusters based on the local density maxima is thus a natural approach.

3.2.3 Temporal Clustering

In the example from the previous section (Fig. 4), spatial clustering has split the two motion patterns into five smaller clusters. Considering solely the spatial relations between clusters is thus insufficient. Temporal clustering takes the smaller motion *clusters* from spatial clustering and combines them to create larger motion *patterns* based on the temporal relations between tracklets.

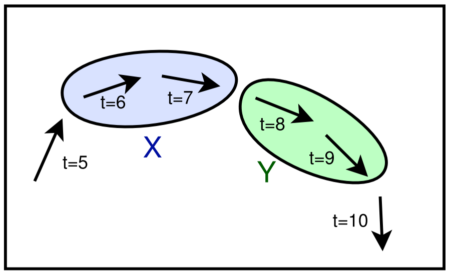


Fig. 5. Example of sequence of tracklets for a node. The arrows represent tracklets while the shaded areas indicate motion clusters.

To motivate this approach, consider Fig. 5 showing the tracklets of one node for $t = 5, \ldots, 10$ and two motions clusters X and Y. When considering the sequence of motion clusters visited by this node, we observe the sequence "$\ldots, X, X, Y, Y, \ldots$". If this sequence also occurs frequently for other nodes, it is a strong indication that the relation "X precedes Y" is strong and these clusters should be concatenated.

Assume spatial clustering yields C clusters $\mathcal{T}_1, \ldots, \mathcal{T}_C$ with each cluster a subset of tracklets (i.e., $\mathcal{T}_c \subseteq \mathcal{T}$) and clusters being disjoint (i.e., $\mathcal{T}_c \cap \mathcal{T}_{c'} = \emptyset$ if $c \neq c'$).

Next, we define $A_{cc'}$ as the *strength* of the temporal relation "$\mathcal{T}_c$ precedes $\mathcal{T}_{c'}$". Let $T_{i,c}$ be the set of *timesteps* for which tracklets by node v_i were assigned to clusters $\mathcal{T}_c$.

$$T_{i,c} = \{t \in T | \tau_{i,t} \in \mathcal{T}_c\}. \tag{4}$$

The entry $A_{cc'}$ is defined as.

$$A_{cc'} = \sum_{v_i \in V} \sum_{t \in T_{i,c}} \sum_{t' \in T_{i,c'}} (1-\gamma)\,\gamma^{t'-t} \mathbb{1}_{t \le t'}. \tag{5}$$

Parameter $\gamma \in [0,1]$ indicates how the influence between clusters decays over time. For example, for $\gamma = 0.9$, the influence decreases by 10 percent for each timestep. The term $\mathbb{1}_{t \le t'}$ indicates that only timestep pairs where $t \le t'$ should be taken into account (i.e., *forward* in time). Note that the total contribution of each tracklet to A is at most 1.

To illustrate this approach, again consider Fig. 5. The behavior of this individual was classified as X for $t = 6, 7$ and as Y for $t = 8, 9$. The strength of the relation from X to Y is thus $\sum_{t=6,7} \sum_{t'=8,9} (1-\gamma)\gamma^{t'-t} = (1-\gamma)(\gamma + 2\gamma^2 + \gamma^3)$.

Exact computation of A is expensive since it requires, for each node v_i, calculating the strength of the relation $\gamma^{t'-t}$ between every pair of tracklets $\tau_{i,t}$ and $\tau_{i,t'}$. Total time complexity for exact computation is thus $\mathcal{O}(nT^2)$ which is infeasible. Instead, we propose a method for approximating A using random samples which has time complexity $\mathcal{O}(nT\kappa)$. Parameter κ is a small constant indicating the number of samples per tracklet (see appendix for details, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2018.2879079).

Finally, we define the cohesion $D(\mathcal{T}_c, \mathcal{T}_{c'})$ between $\mathcal{T}_c$ and $\mathcal{T}_{c'}$ using.

$$D(\mathcal{T}_c, \mathcal{T}_{c'}) = 1 + \frac{A_{cc} + A_{c'c} + A_{cc'} + A_{c'c'}}{|\mathcal{T}_c| + |\mathcal{T}_{c'}|} - \frac{A_{cc}}{|\mathcal{T}_c|} - \frac{A_{c'c'}}{|\mathcal{T}_{c'}|}. \tag{6}$$

TABLE 1
Pipeline Parameters where E.S. Stands for *Expected Sensitivity*

Stage	Param.	Description	E.S.
Proximity Embedding	N	Number of embedding rounds.	Med.
	ϵ	Decrease in learning rate.	Low
	d_{hop}	Average distance per hop.	High
Tracklet Extraction	w	Time window size.	Med.
Spatial Clustering	α	Normalization factor for $\hat{\mathrm{p}}_{i,t}$	High
	β	Normalization factor for $\hat{\mathrm{v}}_{i,t}$	High
	$\delta_{\max}$	Distance threshold for when tracklet is considered cluster center.	Low
	$\rho_{\min}$	Density threshold for when cluster center is considered noise.	Low
Temporal Clustering	γ	Temporal influence between motion patterns.	Low.
	κ	Number of samples per tracklet for approximation of A.	Low
	D_{cut}	Height at which to cut the dendrogram.	High

Detection of motion pattern is done using an hierarchical approach. Initially, each motion pattern corresponds to one motion cluster. Next, the two motion patterns $\mathcal{T}_c$ and $\mathcal{T}_{c'}$ showing the highest score for $D(\mathcal{T}_c, \mathcal{T}_{c'})$ are selected and are merged to create a new motion pattern $\mathcal{T}_{c''} = \mathcal{T}_c \cup \mathcal{T}_{c'}$. This process is repeated until a single pattern remains.

Every time after merging two patterns, A needs to be updated to accommodate the new pattern $\mathcal{T}_{c''}$. This is achieved using the following rules.

$$\begin{aligned} A_{c''x} &= A_{cx} + A_{c'x} \quad &\text{for all} \quad \mathcal{T}_x \neq \mathcal{T}_{c''} \\ A_{xc''} &= A_{xc} + A_{xc'} \quad &\text{for all} \quad \mathcal{T}_x \neq \mathcal{T}_{c''} \\ A_{c''c''} &= A_{cc} + A_{c'c} + A_{cc'} + A_{c'c'}. \end{aligned}$$

As is traditional in hierarchical clustering, the result of this process is a *dendrogram* [11]. The leaves of this tree are the original motion patterns, while the root is a single motion cluster. The final motion patterns are obtained by cutting the dendrogram at a certain height D_{cut}.

4 Analysis

In this section, we discuss the computational complexity of our pipeline and analyze the expected sensitivity of its parameters. Table 1 lists the complete set of parameters of our pipeline.

4.1 Complexity Analysis

We argue our method is scalable based on the computational complexity of each stage.

Proximity Embedding requires repeated execution of SPE for each timestep. The time complexity of SPE is linear in the number of rounds N. In Section 4.2 we will argue that $N = kn$ rounds gives good results, where n is the number of nodes and k is some small integer constant. Repeating SPE for every timestep implies that the time complexity over T timesteps is $\mathcal{O}(knT)$.

Tracklet Extraction requires linear approximation of each subtrajectory of each node at each moment in time. Constructing each tracklet requires $\mathcal{O}(w)$ time and the maximum number of tracklets is nT. The total time complexity is $\mathcal{O}(wnT)$.

Spatial Clustering requires two separate scans over all tracklets: once for determining ρ_s and once for determining δ_s. Both scans require, for each tracklet, querying nearby tracklets. These phases can be accelerated significantly by the use of a space partitioning data structure such as k-d trees [18]. Building a k-d tree takes $\mathcal{O}(nT \log(nT))$ time since there are at most nT tracklets. A single nearest-neighbor query has an average time complexity of $\mathcal{O}(\log(nT))$ of which at most $2nT$ need to be performed. The total time complexity is thus $\mathcal{O}(nT \log(nT))$.

Temporal Clustering consists of two phases: building the similarity matrix A and hierarchical clustering of this matrix. As discussed in Section 3.2.3, exact computation of A is expensive, but an approximation can be constructed in $\mathcal{O}(nT\kappa)$ time. Hierarchical clustering in general has high time complexity of up to $\mathcal{O}(C^3)$ where C is the number of motion clusters. In practice, the actual run-time is negligible since the number of motion clusters is several orders of magnitude lower than the number of tracklets.

In practice, k, w, and κ are small constants which can be ignored since they are negligible compared to the magnitude of nT. We conclude that run-time is dominated by spatial clustering since it has time complexity $\mathcal{O}(nT \log(nT))$.

4.2 Parameter Sensitivity

In this section, we discuss the parameters of our framework and analyze their sensitivity.

Proximity Embedding involves N, ε, and d_{hop}. The number of embedding rounds N per iteration determines a trade-off between quality of the embedding and computational effort: more rounds imply more accurate positions at the cost of additional computation. In practice, we observe that N need not be large for two reasons: (1) nodes locations require only minor adjustments at each timestep and (2) tracklet extraction helps to remove noise from a low-quality embedding. We observed that $N = kn$ often provides a good balance, where k is some small integer constant and n is the number of nodes. For example, in the evaluation (Section 5.1), we obtain good results using $N = 10 \times 50$ for 50 nodes.

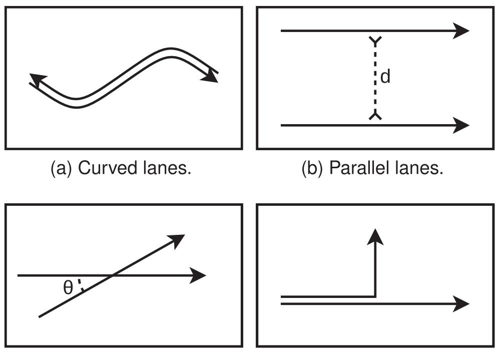
Parameter ϵ has little impact since it exists solely to reduce oscillation. We found $\epsilon = 1 - 0.05^{1/N}$ (i.e., $\lambda = 0.05$ after N rounds) performs well and varying this value has little impact.

The parameter d_{hop} can be set either based on empirical evaluation or theoretical analysis. For example, for the two dimensional case where nodes have an exact detection radius of $d_{\max}$, the average 1-hop distance can be estimated as $\frac{2}{3}d_{\max}$ (see appendix for details, available in the online supplemental material).

Tracklet Extraction involves the window size w which is used for estimating the position vector $\hat{\mathrm{p}}_{i,t}$ and the velocity vector $\hat{\mathrm{v}}_{i,t}$ for a node v_i at time t. A large window helps to remove noise by smoothing the position and velocity, while a small window aims to preserve small-scale structures and “sharp” movements. Note there exists a trade-off between N and w: decreasing N degrades the quality of the embedding, but this can be compensated by increasing w.

Spatial Clustering involves four parameters: $\delta_{\max}$, $\rho_{\min}$, α, and β. We set $\delta_{\max} = 1$ since position and velocity are already normalized by α and β, respectively. Parameter $\rho_{\min}$ is used to discard noise and can be chosen using a ρ-δ scatter plot (see Fig. 4). Such plots should reveal a small number of “noise” points for which δ_s is large but ρ_s is small.

The values of α and β determine the “radius of influence” for each tracklet in terms of position and velocity. As discussed in Section 3.2.2, two tracklets τ_s and τ_r contribute



(a) Curved lanes. (b) Parallel lanes. (c) Intersecting lanes. (d) Divergent lanes.

Fig. 6. Scenarios for synthetic datasets.

to each others density if $\|\hat{p}_s - \hat{p}_r\| \leq \alpha$ and $\|\hat{v}_s - \hat{v}_r\| \leq \beta$. The values of α and β thus determine whether two tracklets belong to the same "class", based on their difference in position (i.e., at most α) and their difference in velocity vector (i.e., at most β).

For example, consider two parallel pedestrian flows at 5 units apart. If $\alpha < 5$, tracklets from different flows do not influence each other and thus two clusters are detected. On the other hand, if $\alpha > 5$, a single cluster could be detected. A similar argument can be made for β when considering two flows having different velocities. The parameters α and β thus must be chosen based on the scenario.

Temporal Clustering involves three parameters: γ, κ, and D_{cut}. The value of κ determines a trade-off between computational effort and loss in quality due to approximation. In general, since the number of tracklets is large (i.e., thousands or millions) and the number of motion clusters is small (i.e., less than 100), we find that the approximation is accurate for small values of κ.

The parameter γ represents how quickly the temporal influence between tracklets degrades after each timestep. Through empirical evaluation we found that values in the range $\gamma \in [0.75, 0.99]$ perform well.

The parameter D_{cut} can be determined by inspecting the resulting dendrogram of temporal clustering. The optimal value for this parameter depends on whether one is interested in coarse-grained patterns (high cut) or fine-grained patterns (low cut).

5 Empirical Evaluation

In this section, we evaluate the performance of our pipeline. We use synthetic models to perform various controlled experiments (Section 5.1) and use real-world datasets to demonstrate the applicability of our method (Section 5.2).

Our prototype[1] is implemented in Python 2.7 and is available under an Open-Source license. We note that our implementation is decently fast. For example, the Hurricane dataset (Section 5.2) generates 7974 tracklets and can be processed in 2.6 seconds on a regular desktop computer.

5.1 Synthetic Dataset

5.1.1 *Experimental Setup*

For the evaluation on synthetic datasets, we consider four different scenarios (Fig. 6) where each is designed to test a different aspect of our processing pipeline. In every scenario, we consider two paths that are followed by simulated pedestrians. Each pedestrian picks a random offset vector of length at most 5 meter, meaning each path can be seen as a "street" or "hallway" which is 10 meter wide.

At the start of the simulation, 25 nodes are positioned at random locations on each path. The walking speed of each node is taken from a normal distribution with a mean of 1.4 m/s and standard deviation of 0.2 m/s, corresponding to the preferred walking speed of humans [19]. Once a node reaches the end of its assigned path, the node is deleted and new node is created at the start of the path. The simulation runs for 1500 timesteps, where each timestep is one simulated second. Each node has a proximity detection radius of 25 meter and anchors are placed along each path every 50 meter.

The simulation output is a proximity graph which is passed the our pipeline. The pipeline generates a set of labeled tracklets, where the labels indicate the motion patterns. We use the normalized mutual information [20] (NMI) score to measure the correlation between the reported labels and the ground-truth labels. The range is between 0 (no correlation) and 1 (perfect correlation).

Unless noted otherwise, parameters are chosen as follows: $N = 500$, $d_{hop} = \frac{2}{3} \times 25$, $\epsilon = 1 - 0.05^{1/N}$, $w = 10$, $\alpha = 15$, $\beta = 0.3$, $\gamma = 0.99$, $\kappa = 25$, and $D_{cut} = 0.5$.

5.1.2 *Scenario A: Curved Lanes*

We consider two opposing sinusoidal paths to test how our pipeline deals with curving flows. Both paths are 250 meter in length and have an amplitude of 50 meter (Fig. 6a)

Fig. 7 shows results for this scenario. Fig. 7a visualizes the resulting trajectories after embedding. Fig. 8 highlights three arbitrary trajectories, showing the amount of noise the pipeline is able to handle.

Fig. 7b shows a density-distance plot of the resulting tracklets. The plot shows that there are several points for which ρ_s is small while δ_s is large. These points could be considered noise and choosing $\rho_{min} = 500$ discards them, removing 0.08 percent of the total tracklets. Spatial clustering discovers 13 motion clusters: 5 moving left-to-right and 8 moving right-to-left. Fig. 7c shows these motion clusters and Fig. 7d shows their centers.

The dendrogram which results from temporal clustering is shown in Fig. 7e The figure reveals that any cut between 0.39 and 0.75 yields two motion patterns. Fig. 7f shows the two motion patterns for $D_{cut} = 0.5$.

The NMI score is 0.963, indicating a high-quality result. We experimented with various amplitudes up to 250 meter and found that this had little impact on the NMI score.

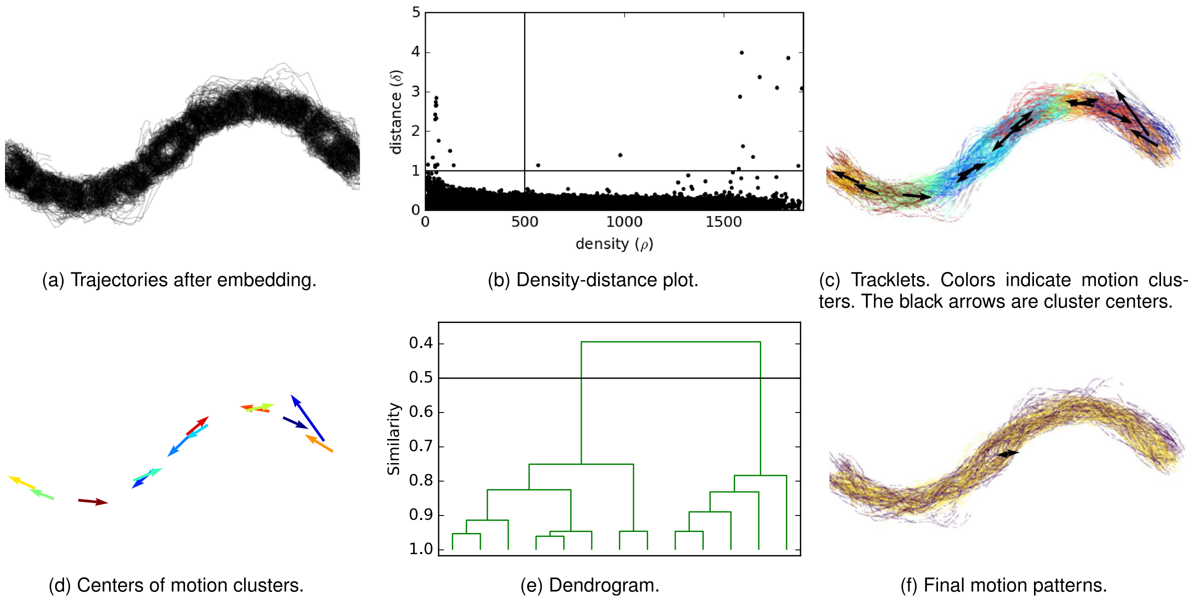
5.1.3 *Scenario B: Parallel Lanes*

Next, we consider two parallel lanes which are 250 meter long and d meter apart (Fig. 6b). This scenario tests the minimal distance required between two lanes in order to separate them.

As discussed in Section 4.2, parameter α plays an important role for this scenario. The value of α needs to be small enough to separate the two lanes, but not too small such that the two lanes themselves are no longer detectable.

Fig. 9 shows a heat map, visualizing the NMI score for various values of α and d. The results show that d needs to be at least 20 meter in order to separate the two lanes, which is a reasonable distance considering the lanes are 10 meter wide.

1. https://github.com/stijnh/scalable-crowd-analysis



(a) Trajectories after embedding. (b) Density-distance plot. (c) Tracklets. Colors indicate motion clusters. The black arrows are cluster centers. (d) Centers of motion clusters. (e) Dendrogram. (f) Final motion patterns.

Fig. 7. Results for Scenario A.

The value of α needs to be at least 10 meter, smaller values fail to detect any lane. The upper bound on α scales linearly with d, meaning far apart lanes become easier to detected.

5.1.4 Scenario C: Intersecting Lanes

Next, we consider the scenario of two crossing lanes (Fig. 6c) which are 250 meter long and intersect at some angle θ, testing the minimum difference in direction required to separate two lanes. The lanes are identical if $\theta = 0^\circ$ and they are perpendicular if $\theta = 90^\circ$.

The value β plays an important role when considering the direction of movement. Fig. 10 shows the NMI score for β versus θ. The figure shows that the two lanes cannot be separated if $\theta \leq 20^\circ$. For $\theta \geq 50^\circ$, the results show high NMI scores meaning the two lanes are correctly detected. An interesting case is for $20^\circ < \theta < 50^\circ$, since the results show decent NMI scores (around 0.5) but only if $0.15 \leq \beta \leq 0.3$. If β is too small, many tracklets are labeled as noise since their density falls below threshold $\rho_{\min}$. If β is too large, it is impossible to separate the two lanes. The second problem is demonstrated in Fig. 11, showing spatial clustering cannot distinguish two clusters at the intersection point if β is large.

We note that the reason lanes cannot be separated for $\theta \leq 20^\circ$ is mostly due to the noise introduced by the embedding. More accurate positioning would decrease this minimum angle.

Fig. 8. Three examples of trajectories from Fig. 7a.

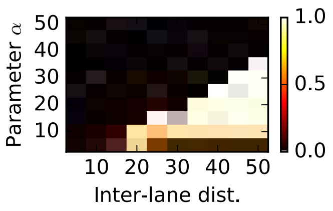


Fig. 9. NMI-scores for Scenario B.

5.1.5 Scenario D: Divergent Lanes

Finally, we test our pipeline on diverging flows by considering a T-intersection (Fig. 6d) with two lanes: one straight lane and one lane containing a bend at the mid-point. Our pipeline should detect three patterns: one for each arm of the T-intersection.

Fig. 12 shows the results for this scenario. Spatial clustering detects 8 motion clusters, see Fig. 12a. The resulting dendrogram (Fig. 12b) shows that any cut between 0.75 and 0.92 yields three motion patterns. Fig. 12c shows that these patterns are the desired outcome.

We note that a scenario with *converging* lanes instead of divergent lanes would give the same result, since the datasets would be identical with the only difference being that the velocity vectors are inverted.

5.2 Real-World Dataset

In this section, we evaluate the applicability of our approach to real-world data. We show results for two datasets: a hurricane track dataset and an animal movement dataset. These datasets were previously used by Lee et al. [21] for evaluation of their sub-trajectory clustering algorithm, enabling

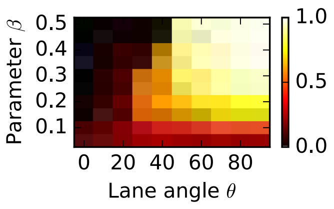
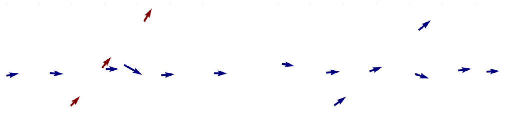
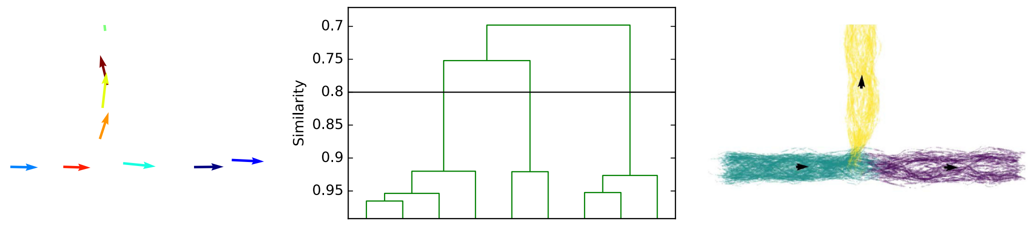


Fig. 10. NMI-scores for Scenario C.



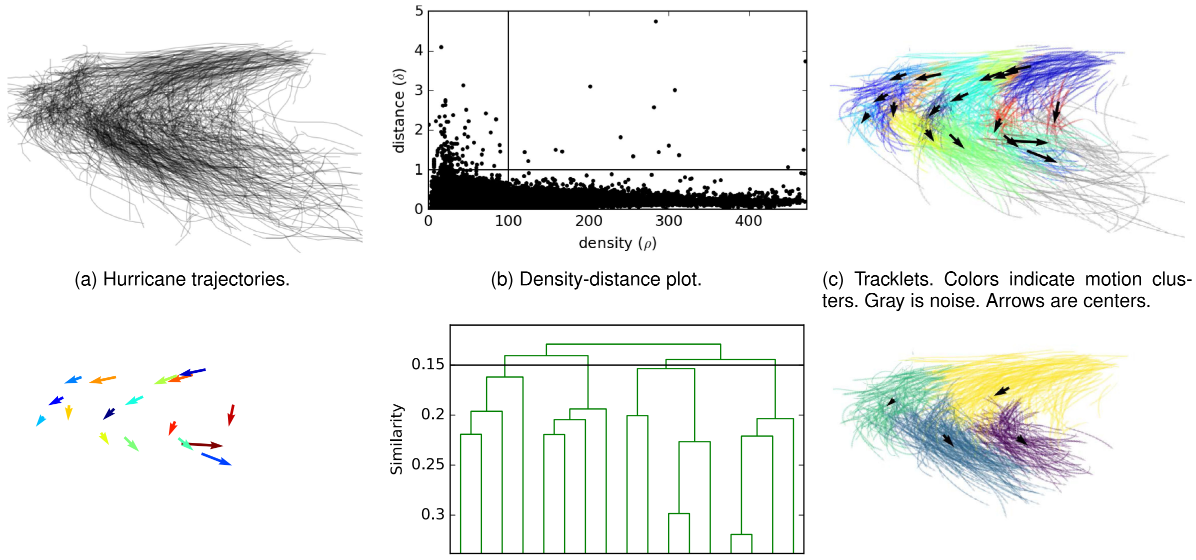
(a) Two lanes for $\beta = 0.3$. (b) One lane for $\beta = 0.5$.

Fig. 11. Motion cluster centers for Scenario C ($\theta = 40^\circ$). At the intersection, either one center ($\beta = 0.5$) or two centers ($\beta = 0.3$) are detected.



(a) Centers of motion clusters. (b) Dendrogram. (c) Final motion patterns

Fig. 12. Results for Scenario D.



(a) Hurricane trajectories. (b) Density-distance plot. (c) Tracklets. Colors indicate motion clusters. Gray is noise. Arrows are centers.

(d) Centers of motion clusters. (e) Dendrogram. (f) Final motion patterns.

Fig. 13. Results for hurricane dataset.

direct comparison between our work and their method. Since these datasets provide absolute coordinates, we omit the embedding within our pipeline and focus on motion pattern detection.

5.2.1 Hurricane Track Data

The hurricane "best track" dataset[2] contains the latitude and longitude of important Atlantic hurricanes at 6-hourly interval from the years 1950 through 2004. The data set consists of 570 trajectories and 17,736 points.

Fig. 13a visualizes the hurricane trajectories taken from the dataset. Spatial clustering was performed for $w = 12$ hours, $\alpha = 300$ km, and $\beta = 40$ km/hour. Fig. 13b shows a density-distance plot, revealing that this dataset contains some noise as indicated by the few points for which δ_s is large and ρ_s is small. Selecting $\rho_{\min} = 100$ removes the noise, discarding 8.3 percent of the total tracklets.

Fig. 13c shows the remaining tracklets and Fig. 13d shows the centers of the detected motion clusters. Temporal clustering was performed for $\gamma = 0.75$, see Fig. 13e for the resulting dendrogram. The cut was performed at $D_{\text{cut}} = 0.87$.

Fig. 13f shows the final four motion patterns detected on the hurricane dataset. This figure reveals that the dataset contains a clear dominant "curvature": hurricanes originate from the north-east, move towards the south-west, and terminate either in the west or in the south-east.

5.2.2 Animal Movement Data

The animal movement dataset originates from the Starkey project[3]. The dataset contains radio-telemetry data for various wildlife animals in the Starkey nature reserve from 1993 through 1996. Each data record corresponds to one measurement and consists of the animal's identifier, the animal's species, time, and absolute coordinates.

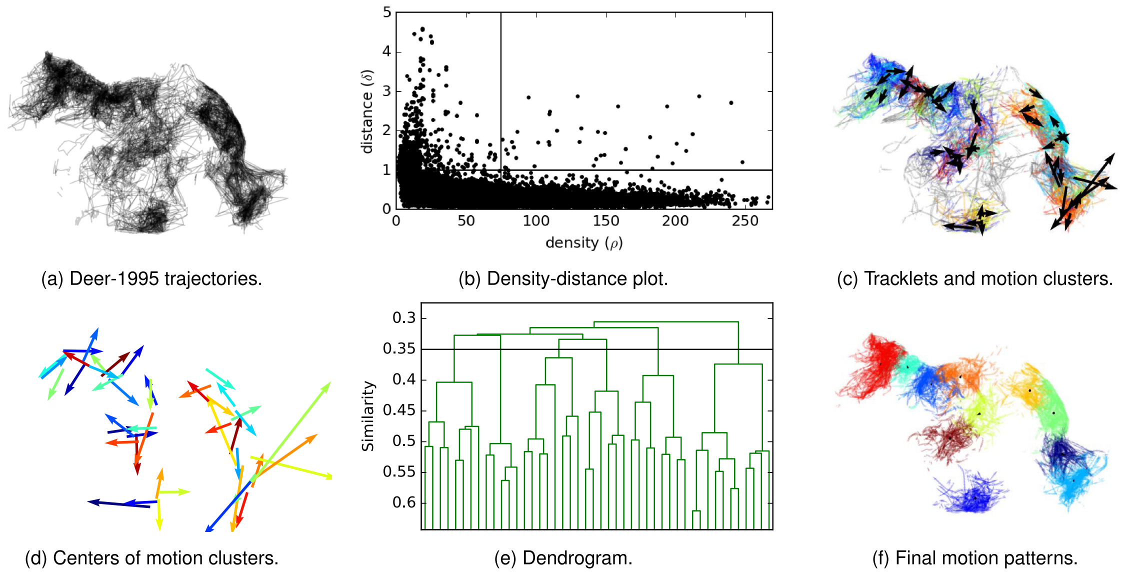
The data is not recorded with any regular interval, a requirement for our method. We preprocessed the dataset by resampling the positions at an interval of one hour, estimating the position by interpolating between the pair of closest measurements. The dataset also contains many "gaps" where no data is available for an animal for several hours or even days. We divide the data for each animal into multiple trajectories based on gaps of 3 hours or longer.

Fig. 14a visualizes the trajectories for deer in 1995 (32 animals, 50505 points). Fig. 14b shows the results of spatial clustering for $w = 3$ hours, $\alpha = 0.5$ km and $\beta = 0.1$ km/h. Comparing Figs. 13b and 14b shows that the animal dataset contains more noise than the hurricane dataset, indicated by the large number of tracklets for which δ_s is large and ρ_s is small. Based on this plot, $\rho_{\min} = 75$ was selected which discards 15.5 percent of the tracklets.

Figs. 14c and 14d visualize the remaining tracklets and resulting motion clusters. Comparing the motion pattern centers in Figs. 13d and 14d reveals that the movement of deer is much more chaotic and unstructured than the

2. http://weather.unisys.com/hurricane/atlantic

3. https://www.fs.fed.us/pnw/starkey/



(a) Deer-1995 trajectories. (b) Density-distance plot. (c) Tracklets and motion clusters. (d) Centers of motion clusters. (e) Dendrogram. (f) Final motion patterns.

Fig. 14. Results for Animal Movement dataset.

movement of hurricanes. This is expected since animals wander randomly without a clear direction, while the hurricanes all follow a similar path.

Temporal clustering was performed for $\gamma = 0.75$, see Fig. 14e for the resulting dendrogram. Fig. 14f shows the motion patterns when performing the cut at $D_{\text{cut}} = 0.65$. The figure shows that each motion pattern corresponds to a certain "region" of the nature reserve, but there is no dominant direction of motion within each region.

When considering the individual trajectories of deer (see Fig. 15), we can conclude that deer mostly stay within certain regions and rarely cross boundaries between regions. The detected motion patterns are meaningful since they closely match these regions from the original dataset. One possible explanation for this behavior is that deer are territorial animals, each protecting its own territory. Another explanation is that there are physical boundaries between the regions (i.e., rivers, roads, fences, hills, etc.).

5.2.3 *Comparison against Existing Work*

We compare our results against those of TRACLUS [21], which is commonly used for trajectory clustering [22], [23], [24], [25], [26] (Section 6). TRACLUS partitions each trajectory into a series of line segments, clusters the set of segments, and calculates a "representative path" for each cluster. For evaluation, the implementation of TRACLUS in Java was used[4]. The parameters $(MinLns, \varepsilon)$ were taken from the original manuscript on TRACLUS [21].

Fig. 16a shows results for the hurricane dataset ($\varepsilon = 30$, $MinLns = 6$). TRACLUS detects eight clusters: two large and six small clusters. The large clusters are straight lines: one from the north-east to the west and one from the west to the south-east. Comparing our results (Fig. 13f) to TRACLUS' (Fig. 16a) shows that only our method was able to capture the "curvature" of the trajectories.

Fig. 16b shows results for the Deer-1995 dataset ($\varepsilon = 29$, $MinLns = 8$). TRACLUS has detected two elongated clusters: one along the north-side and one along the east-side of the nature reserve. However, previously we saw that deer mostly wander randomly and stay within certain territorial regions (Fig. 15). The representative trajectories discovered by TRACLUS do not represent the original dataset and seem to be the result of concatenating multiple sub-trajectories of different animals. Our method, on the other hand, was able to detect meaningful motion patterns (Fig. 14f).

6 Related Work

To the best of our knowledge, our method is the first complete end-to-end solution for extracting motion patterns

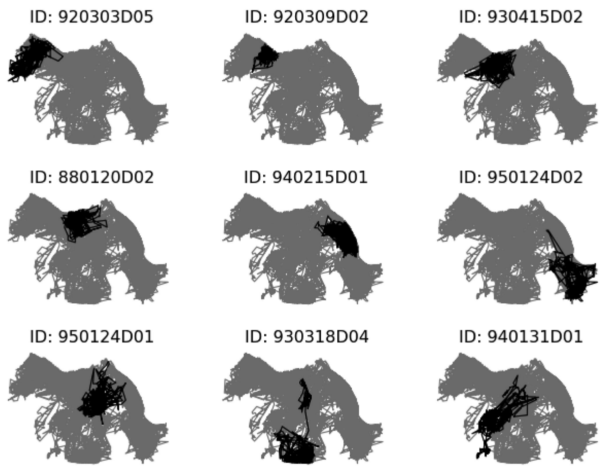


Fig. 15. Trajectories of nine animals from the deer-1995 dataset. ID is the unique animal identifier assigned by the Starkey nature reserve.

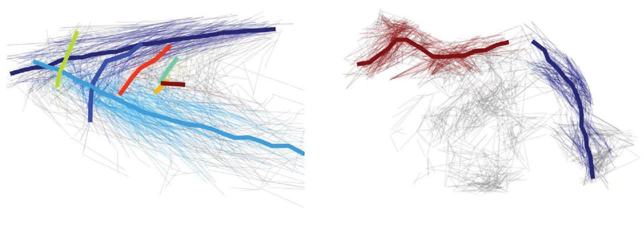

(a) Hurricane dataset. (b) Deer-1995 dataset.

Fig. 16. Results of the TRACLUS algorithm. Different colors indicate different clusters. Bold lines are the representative trajectories.

4. https://github.com/luborliu/TraClusAlgorithm

from real-world crowd based on proximity data. However, crowd analysis is a popular research topic in many different domains. In this section, we discuss relevant contributions from three areas of research: proximity sensing, computer vision, and data mining.

6.1 Proximity Sensing

Crowd analysis by using proximity sensors has proven to be a promising area of research. Martella et al. showed how proximity graphs could be used for analyzing social interactions at an IT conference [3], positioning people in a six-story building using only a handful of anchor points [27], and clustering the paths of museum visitors [5]. However, further research has been scarce.

6.2 Computer Vision

The analysis of crowds is also an active research topic in computer vision. For a comprehensive overview of all crowd-related work from computer vision, we refer to the excellent literature reviews by Zhan et al. (2008) [28], Thida et al. (2013) [29], Li et al. (2015) [30], and Grant et al. (2017) [31]. Most of this work focuses on automatic analysis of videos from surveillance cameras. Many methods have been proposed, for example, to understand crowd behavior, track crowd members, or estimate crowd density.

One particular topic which is related to our method is *crowd flow segmentation* [30]: the problem of diving the camera view into regions of coherent motions. For example, Ali et al. [32] show how techniques from computational fluid dynamics are suitable for crowd segmentation based on the intuition that high-density crowds behave similar to fluids. Hu et al. [33] presented a method which detects motion flow vectors and groups these into clusters using a hierarchical agglomerative clustering algorithm. Benabbas et al. [34] followed an approach which divides the camera view into rectangular blocks, detects the dominant motion vectors within each block, and clusters adjacent blocks containing similar motions. Zhao & Medioni [35] describe a method which tracks moving objects, extracts tracklets from these trajectories, and embeds these points into (p_x, p_y, θ) space (p is position, θ is direction). These points form intrinsic manifold structures which are segmented using a robust manifold grouping algorithm.

Using proximity sensors reveals several advantages over cameras. First, proximity sensing allows holistic analysis of large areas, such as music festivals, stadiums or musea. Cameras are inherently limited to one single static perspective and there seems little research on how to combine the results from multiple cameras. Second, our method focuses on scalability, allowing it to be used for analyzing the movements of massive crowd over long periods of time. Computer vision algorithms are often complex and computationally expensive. Third, the majority of these methods segment the camera view into disjoint regions, meaning they cannot handle “overlapping” flows while our method can handle these cases.

6.3 Data Mining

Due to the developments in mobile computing and location-acquisition device, there has recently been much research in data mining of trajectories from moving entities, such as humans, animals, or vehicles. For a complete overview of work in trajectory data mining, we refer to the extensive surveys by Zheng (2015) [22], Feng & Zhu (2016) [36], and Mazimpaka et al. (2016) [37].

The second stage of our pipeline could be seen as a clustering problem: given a set of trajectories, group “popular” subtrajectories into clusters. According to Zheng [22], there are three approaches to trajectory clustering.

The first approach is to define a similarity metric for trajectories and group them using traditional clustering algorithms. For example, Morris & Trivedi [38] performed an in-depth evaluation of this idea for various metric, algorithms, and datasets. However, this approach treats trajectories as atomic units and captures movements patterns only if individuals travel together simultaneously. As such, it is not suitable for our pipeline.

The second approach is to project trajectories onto a map (e.g., road network) and employ graph algorithms to find popular paths (i.e., subgraphs). For example, NETSCAN [39] is an algorithm based on this idea of detecting “hot” paths in graphs. While this approach works well for vehicles which are constraint to roads, it is not suitable for people in open spaces.

The third approach is a *micro-macro* framework [22]. These methods first partition the trajectories into sets of short subtrajectories (micro-level), and then group these segments into clusters (macro-level). The de facto standard algorithm in this category is TRACLUS [21]. TRACLUS segments each input trajectory into a series of line segments, according to the Minimum Description Length principle, and then clusters the complete set of line-segments using a density-based clustering algorithm. TRACLUS (and slight variations) have been used for many different purposes including trajectory classification [23], trajectory outlier detection [24], analysis of animal movement [25], and discovery of popular traffic routes [26].

There are key differences between our pattern detection method and TRACLUS. First, since TRACLUS relies on clustering of straight line segments, it cannot handle sharp turns (demonstrated in Fig. 16a). Our method can handle these cases since it takes temporal relations into account. Second, TRACLUS clusters the line segments without considering their “owner”, meaning it suffers from the problem of concatenating different sections from different nodes (demonstrated in Fig. 16b). Third, TRACLUS does not take velocity of objects into account, meaning it cannot differentiate objects of different velocities on the same road (e.g., cyclists and cars). Our method explicitly takes velocity into account.

7 Conclusions & Future Work

In this work, we presented a complete end-to-end processing solution for detecting motion patterns in real-world crowds. Our method is designed to be fast and resilient against noise, allowing it to be used for large real-world crowds. For measuring crowd movements, we utilize proximity graphs followed by a fast embedding algorithm. For detection of patterns, we designed a three-stage procedure which considers spatial and temporal relations separately. Results show that our method works well, both on synthetic simulations and real-world datasets.

For future work, we are exploring methods for automatically tuning the parameter α, β, and D_{cut} based on the datasets. We are also considering methods for including time-ofday into motion pattern detection. For example, motion pattern "enter office" would be popular in the morning and "exit office" during the evening. Furthermore, we are designing an incremental variation of our framework, allowing new data to be added without re-executing the entire pipeline. This would allow for a system which receives and processes data from proximity sensors in real-time. Finally, we are working on obtaining real-world proximity measurements to evaluate our method on non-synthetic proximity datasets.

Overall, we consider proximity graphs to be a promising method for the analysis of crowds and we see our work as a first step towards rich crowd pattern analysis.

REFERENCES

[1] C. Castellano, S. Fortunato, and V. Loreto, "Statistical physics of social dynamics," *Rev. Modern Phys.*, vol. 81, pp. 591–646, 2009.

[2] C. Martella, M. van Steen, A. Halteren, C. Conrado, and J. Li, "Crowd textures as proximity graphs," *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 114–121, Jan. 2014.

[3] C. Martella, M. Dobson, A. van Halteren, and M. van Steen, "From proximity sensing to spatio-temporal social graphs," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2014, pp. 78–87.

[4] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global Positioning System: Theory and Practice*. Berlin, Germany: Springer, 2012.

[5] C. Martella, A. Miraglia, M. Cattani, and M. van Steen, "Leveraging proximity sensing to mine the behavior of museum visitors," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2016, pp. 1–9.

[6] S. Liu, Y. Jiang, and A. Striegel, "Face-to-face proximity estimationusing bluetooth on smartphones," *IEEE Trans. Mobile Comput.*, vol. 13, no. 4, pp. 811–823, Apr. 2014.

[7] S. G. Kobourov, "Spring embedders and force directed graph drawing algorithms," *CoRR*, vol. abs/1201.3011, 2012, http://arxiv.org/abs/1201.3011

[8] I. Borg and P. J. Groenen, *Modern Multidimensional Scaling: Theory and Applications*. Berlin, Germany: Springer, 2005.

[9] D. K. Agrafiotis, "Stochastic proximity embedding," *J. Comput. Chemistry*, vol. 24, no. 10, pp. 1215–1221, 2003.

[10] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A K-means clustering algorithm," *J. Roy. Statistical Soc. Series C (Appl. Statist.)*, vol. 28, no. 1, pp. 100–108, 1979.

[11] L. Rokach and O. Maimon, "Clustering methods," in *Data Mining and Knowledge Discovery Handbook*. Berlin, Germany: Springer, 2005, pp. 321–352.

[12] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 8, pp. 790–799, Aug. 1995.

[13] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *Comput. J.*, vol. 16, no. 1, pp. 30–34, 1973.

[14] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, vol. 96, pp. 226–231.

[15] A. Vedaldi and S. Soatto, "Quick shift and kernel methods for mode seeking," in *Proc. Eur. Conf. Comput. Vis.*, 2008, pp. 705–718.

[16] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Sci.*, vol. 344, no. 6191, pp. 1492–1496, 2014.

[17] W. L. G. Koontz, P. M. Narendra, and K. Fukunaga, "A graph-theoretic approach to nonparametric cluster analysis," *IEEE Trans. Comput.*, vol. C-25, no. 9, pp. 936–944, Sep. 1976.

[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[19] B. J. Mohler, W. B. Thompson, S. H. Creem-Regehr , H. L. Pick, and W. H. Warren, "Visual flow influences gait transition speed and preferred walking speed," *Exp. Brain Res.*, vol. 181, no. 2, pp. 221–228, 2007.

[20] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *J. Mach. Learn. Res.*, vol. 11, pp. 2837–2854, 2010.

[21] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: A partition-and-group framework," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2007, pp. 593–604.

[22] Y. Zheng, "Trajectory data mining: an overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, 2015, Art. no. 29.

[23] J.-G. Lee, J. Han, X. Li, and H. Gonzalez, "Traclass: Trajectory classification using hierarchical region-based and trajectory-based clustering," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 1081–1094, 2008.

[24] J.-G. Lee, J. Han, and X. Li, "Trajectory outlier detection: A partition-and-detect framework," in *Proc. IEEE 24th Int. Conf. Data Eng.*, 2008, pp. 140–149.

[25] Z. Li, M. Ji, J.-G. Lee, L.-A. Tang, Y. Yu, J. Han, and R. Kays, "Movemine: Mining moving object databases," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2010, pp. 1203–1206.

[26] X. Li, J. Han, J.-G. Lee, and H. Gonzalez, "Traffic density-based discovery of hot routes in road networks," in *Proc. Int. Symp. Spatial Temporal Databases*, 2007, pp. 441–459.

[27] C. Martella, M. Cattani, and M. van Steen, "Exploiting density to track human behavior in crowded environments," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 48–54, Feb. 2017.

[28] B. Zhan, D. N. Monekosso, P. Remagnino, S. A. Velastin, and L.-Q. Xu, "Crowd analysis: A survey," *Mach. Vis. Appl.*, vol. 19, pp. 345–357, 2008.

[29] M. Thida, Y. L. Yong, P. Climent-Pérez, H.-l. Eng, and P. Remagnino, "A literature review on video analytics of crowded scenes," in *Intelligent Multimedia Surveillance*. Berlin, Germany: Springer, 2013, pp. 17–36.

[30] T. Li, H. Chang, M. Wang, B. Ni, R. Hong, and S. Yan, "Crowded scene analysis: A survey," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 3, pp. 367–386, Mar. 2015.

[31] J. M. Grant and P. J. Flynn, "Crowd scene understanding from video: a survey," *ACM Trans. Multimedia Comput., Commun. Appl.*, vol. 13, no. 2, 2017, Art. no. 19.

[32] S. Ali and M. Shah, "A lagrangian particle dynamics approach for crowd flow segmentation and stability analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2007, pp. 1–6.

[33] M. Hu, S. Ali, and M. Shah, "Learning motion patterns in crowded scenes using motion flow field," in *Proc. 19th Int. Conf. Pattern Recognit.*, 2008, pp. 1–5.

[34] Y. Benabbas, N. Ihaddadene, and C. Djeraba, "Motion pattern extraction and event detection for automatic visual surveillance," *EURASIP J. Image Video Process.*, vol. 2011, 2011, Art. no. 7.

[35] X. Zhao and G. Medioni, "Robust unsupervised motion pattern inference from video and applications," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2011, pp. 715–722.

[36] Z. Feng and Y. Zhu, "A survey on trajectory data mining: Techniques and applications," *IEEE Access*, vol. 4, pp. 2056–2067, 2016.

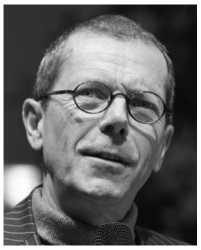
[37] J. D. Mazimpaka and S. Timpf, "Trajectory data mining: A review of methods and applications," *J. Spatial Inf. Sci.*, vol. 2016, no. 13, pp. 61–99, 2016.

[38] B. Morris and M. Trivedi, "Learning trajectory patterns by clustering: Experimental studies and comparative evaluation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 312–319.

[39] A. Kharrat, I. S. Popa, K. Zeitouni, and S. Faiz, "Clustering algorithm for network constraint trajectories," in *Headway in Spatial Data Handling*. Berlin, Germany: Springer, 2008, pp. 631–647.

Stijn Heldens received the MSc degree in computer science from VU University Amsterdam. His research interests lie in data mining, parallel algorithms, and high-performance computing. He was previously a researcher at the Digital Society Institute at the University of Twente and is currently an eScience research engineer at the Netherlands eScience Center.

Nelly Litvak received the PhD degree in stochastic operations research from the Eindhoven University of Technology. She is a professor with the Applied Mathematics Department, University of Twente, and (part-time) at the Department of Mathematics and Computer Science, Eindhoven University of Technology. Her research interests include complex networks, algorithms, stochastic models, and random graphs. She is a managing editor of *Internet Mathematics* and associate editor of *Stochastic Processes and Their Applications*. She has been on program committees of the World Wide Web, KDD, and INFORMS Applied Probability conferences.


Maarten van Steen is a professor with the University of Twente, where he is the scientific director of the Digital Society Institute. He is specialized in large-scale distributed systems, now concentrating on very large wireless distributed systems, notably in the context of crowd monitoring using gossip-based protocols for information dissemination. Next to Internet-based systems, he has published extensively on distributed protocols, wireless (sensor) networks, and gossiping solutions. He is associate editor for *IEEE Internet Computing*, field editor for *Springer Computing*, and section editor for *Advances in Complex Systems*. He authored and co-authored three textbooks, including *Distributed Systems* (with Andrew Tanenbaum), now in its 3rd edition, as well as an *Introduction to Graph Theory and Complex Networks*. He is a senior member of the IEEE.