FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. David Říha

Group Detection in Crowds Using Spatiotemporal Data

Computer Science Institute of Charles University

Supervisor of the master thesis: Ing. David Hartman, Ph.D.

Study program: Computer Science
Specialization: Artificial Intelligence
Prague 2024

I declare that I carried out this master thesis independently and only with the cited sources, literature, and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague date 11. January 2024 David Říha

I would like to thank my supervisor, David Hartman, for his help and guidance. I also thank Giulio Tani Raffaelli for consultations and Simona Kurňavová for read through.

Title: Group Detection in Crowds Using Spatiotemporal Data

Author: Bc. David Říha

Department / Institute: Computer Science Institute of Charles University

Supervisor of the master thesis: Ing. David Hartman, Ph.D.

Abstract: This thesis addresses the challenge of social group detection in crowds, presenting an algorithm informed by sociological insights into common group formations among pedestrians. Our proposed algorithm demonstrates comparable performance to existing solutions – Time-sequence DBSCAN and Agglomerative Hierarchical Clustering with Hausdorff Distance, using the DIAMOR dataset for testing and comparison. Additionally, we introduce a validator tool potentially capable of refining results from existing algorithms based on a group shape criterion, leading to improved accuracy in identifying groups.

Keywords: groups detection; clustering; group shape analysis; pedestrian behavior;

Table of Contents

Introduction

The study of crowd behavior extends across various fields such as psychology, sociology, and informatics, offering insights into human behavior. The knowledge gained can be instrumental in various applications, ranging from enhancing safety measures to innovating crowd management strategies and designing spaces accommodating large human flows.

In this thesis, we analyze and develop methodologies to identify social groups using spatiotemporal data, aiming to contribute to the field by proposing a novel algorithm for solving this problem. Our proposed algorithm utilizes findings of crowd behavior studies, from which we derive a new measure called group shape score. The thesis involves a comparison of our solution with two existing widely used approaches to social group detection, namely Time-sequence DBSCAN [1] and Agglomerative Hierarchical Clustering with Hausdorff Distance [2]. The evaluation is conducted using the DIAMOR dataset [3], which contains ground truth data obtained by human observations, providing a benchmark for comparing our results.

Additionally, we introduce a group shape score validator as a post-processing mechanism to refine the results obtained by any algorithm addressing this problem. This validator employs a group shape score in a similar manner as our proposed algorithm. We test this on both selected existing approaches to assess its effectiveness in enhancing the precision of group detection results.

Acknowledging ethical considerations is crucial in this area of crowd behavior analysis due to privacy concerns. It is vital that the insights gained from this discipline are utilized ethically, and we emphasize the importance of collective responsibility within the context of this thesis to prevent any misuse or harmful actions.

1 Problem description

In this chapter, we present the problem of group detection among pedestrians. The significance of addressing this challenge extends to various practical applications, including but not limited to crowd management and behavioral analysis. The ability to differentiate and categorize groups of people within a given environment holds the potential for advancing decision-making systems in complex scenarios.

1.1 Social Groups

The definition of social groups is not consistent across different areas of expertise and scientific studies. A study by Jarosław Was and Krzysztof Kulakowski defines it as "A group in a crowd is interpreted as two or more persons who are connected by interpersonal relationships" [4]. For the purpose of this thesis, we shall define social groups as consisting of two or more individuals intentionally walking together. Such groups usually consist of family members, friends, or colleagues. These two definitions do not necessarily coincide with each other in all cases.

The difficulty of a group detection task from tracking data is apparent when we try to translate the definition of social groups to a strict mathematical or logical formulation. For example, while walking closely together could be a good indicator of pedestrians forming a social group, in many situations, pedestrians share a common goal and also a common walking path without forming a social group. In public transportation like the subway, it is common to have a limited number of exits and entrances that every passenger can walk through. In such situations, many passengers share the same start and end point with very few available options for their path. In this situation grouping people only by their proximity to each other might prove inaccurate, especially during high-density situations.

Due to the reasons mentioned above another problem arises when observing social groups in video footage of pedestrians. The difficulty lies not only in translating this problem to mathematical representation but also in the inherent subjectivity of visual interpretation of the footage. Unless prior knowledge about observed individuals is available, it is a difficult task to distinguish between actual social groups and people just sharing a similar path to their destination and,

therefore, obtain ground truth about given video footage. For simplification, we assume ground truth data provided by human observation are true, but keeping in mind the possibility of human error in this interpretation.

1.2 Data for Group Detection

The approach to group detection varies based on the characteristics of the chosen dataset, its representation, and pre-processing. Some studies choose to work with video data of pedestrians. This allows researchers to observe and categorize human interactions such as verbal communication, head orientation, hand gestures, etc. Since this task is inherently based on social interactions, the usage of video footage can prove to be advantageous.

In this thesis, we will be using two-dimensional temporal trajectories, without any other additional data. This choice limits us to position and movement-based algorithms.

Definition (Trajectory): For each pedestrian p we define their ***trajectory*** *as*

$$T_p = \left[\left(x_{t_0}, y_{t_0} \right), \left(x_{t_1}, y_{t_1} \right), \cdots, \left(x_{t_n}, y_{t_n} \right) \right],$$
$$t_i < t_j \Rightarrow i < j$$

where $\left(x_{t_i}, y_{t_i} \right) \in R^2$ are spatial coordinates at time $t_i \in R$. Trajectory T_p at time t will be denoted as $T_p[t] = \left(x_t, y_t \right) \in T_p$. In the rest of the thesis, the terms *pedestrian* and (their) *trajectory* are used interchangeably.

1.3 Reduction to Clustering

Clustering is the process of organizing a collection of items in a manner where items within the same cluster share more similarities than those in different clusters. Clustering aims to uncover natural groupings or patterns present in a dataset. Objects or data points that exhibit common characteristics (commonly described as being "closer to each other") are grouped together, forming clusters. This method aids in revealing inherent structures within data, making it a valuable tool for tasks such as identifying similar customer behaviors, categorizing images, or organizing information. It was first introduced by Driver and Kroeber in 1932 [5].

Definition (Partition of Set): Let T be a set. ***Partition of set*** T is a non-empty collection $C=\{C_1, C_2, \cdots, C_m\}$ such that:

- $\forall i : C_i \subseteq T$,
- $\forall i, j : C_i \cap C_j = \varnothing$,
- $\cup_i C_i = T$.

Definition (Clustering): Let $T=\{T_1, T_2, \cdots, T_n\}$ be a set of objects and $C=\{C_1, C_2, \cdots, C_m\}$ is a partition of set T, then we say that C is a ***clustering*** of T and C_i are called ***clusters***. We can also say that function $f : T \to \{1, 2, \cdots, m\}$, where $f(T_i) = j$ indicates that $T_i \in C_j$ describes ***clustering***.

The group detection problem can be looked at as clustering. Each pedestrian represents an object that we want to categorize into a group (cluster) with other pedestrians that are 'similar', using its spatiotemporal data. Characteristics, according to which decision is made if spatiotemporal data are 'similar' and should belong to the same group, will be described later in the chapter 5. No pedestrian can be in more than one group. Formally definition for group detection problem:

Definition (Group Detection Problem): Given a set of n trajectories $T=\{T_1, T_2, \cdots, T_n\}$ and ground truth clustering G_{gt} of T. Find groups $\{G_1, G_2, \cdots, G_m\}$ such that $\{G_1, G_2, \cdots, G_m\} = G_{\text{gt}}$.

1.4 Evaluation of Clustering

To compare the results of different algorithms, we will use the Adjusted Rand Index (ARI) as our evaluation function. ARI evaluates the similarity between two clustering assignments, ignoring permutations. It provides a score that indicates how well the clustering results align with each other, accounting for the possibility of randomness.

We are going to use ARI as an evaluation function that quantifies the agreement between the true groupings of data points and the clusters computed by a clustering algorithm. The "adjusted" part considers what would be expected by random chance, providing a normalized measure that ranges from the worst value of -1 to the best

value of 1. Clusterings for which ARI is close to 0 can be interpreted as if produced by a random algorithm, and any value of ARI lower than 0 suggests that the algorithm by which it was produced is performing worse than the random one [6].

Definition (Adjusted Rand Index): Given a set of trajectories $T=\{T_1,T_2,\cdots,T_n\}$, and clusterigns $X=\{X_1,X_2,\cdots,X_r\}$, $Y=\{Y_1,Y_2,\cdots,Y_s\}$ of *T,* then ARI can be calculated as:

$$ARI(X,Y)=\frac{\sum_{ij}\left(\frac{n_{ij}}{2}\right)-\left[\sum_i\left(\frac{a_i}{2}\right)\sum_j\left(\frac{b_j}{2}\right)\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_i\left(\frac{a_i}{2}\right)+\sum_j\left(\frac{b_j}{2}\right)\right]-\left[\sum_i\left(\frac{a_i}{2}\right)\sum_j\left(\frac{b_j}{2}\right)\right]/\binom{n}{2}},$$

where $n_{ij}=|X_i\cap Y_j|$, $\forall X_i\in X: a_i=\sum_{j=1}^{s} n_{1j}$, and $\forall Y_j\in Y: b_j=\sum_{i=1}^{r} n_{i1}$.

ARI serves as an evaluation function of clusterings. Algorithms approximate solution of group detection problem by producing a set of groups that are similar to the groups from the ground truth. The similarity is evaluated by ARI.

Definition (Approximation of Group Detection Problem): Given a set of trajectories $T=\{T_1,T_2,\cdots,T_n\}$ and ground truth clustering G_{gt}, find clustering $G=\{G_1,G_2,\cdots,G_m\}$ such that $ARI(G,G_{\text{gt}})$ is maximalized.

2 Existing methods

In this chapter, we focus on understanding methods of social group detection, emphasizing two techniques as benchmarks for our proposed algorithm's evaluation: Time-sequence DBSCAN and Agglomerative Hierarchical Clustering with Generalized Hausdorff Distance.

We aim to outline and examine these methods, considering their strengths and limitations. This forms the basis for evaluating their performance and comparing it with our suggested solution. This assessment is essential for developing our own group detection algorithms, providing insights and a potential roadmap for future improvements.

The selection of Time-sequence DBSCAN was motivated by its great performance, as reported in the respective research paper. The authors compared three models that considered overlapping coexisting time, distance between trajectories, or both, demonstrating notably positive results. The evaluation encompassed three datasets from the German university area at various times of the day, and the algorithm exhibited robust performance [1].

Agglomerative Hierarchical Clustering with Generalized Hausdorff Distance was chosen for its contrasting approach compared to Time-sequence DBSCAN. This clustering method draws inspiration from psychological models of collective behavior. The algorithm underwent testing on diverse data sets, incorporating indoor, outdoor, and varying viewpoints [2].

2.1 Time–Sequence DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was presented by Martin Ester, Hans-Peter Kriegel, Jiirg Sander, and Xiaowei Xu in 1996 [7]. It belongs to the group of clustering algorithms used for class identification in spatial databases. According to their paper, DBSCAN excels in large spatial databases, offering benefits such as minimal domain knowledge requirements for input parameters, the discovery of clusters with arbitrary shapes, and efficient processing even in databases with a higher number of objects.

DBSCAN uses the concept of *core points* – points with a high density of points around them – for cluster creation and *border points* – points that are reachable from core points – to distinguish between noise and points that are part of some cluster. It takes two parameters as input:

1. *epsilon* – defines the radius of a circle around points for conducting a range query, establishing whether other points are considered "close" (located inside the circle) or not
2. *minPoints* – defines the minimum number of points required to start a cluster

DBSCAN is not a deterministic algorithm since border points may be associated with multiple clusters, and their assignment to the cluster depends on the core point from which they are first identified.

Time-sequence DBSCAN, introduced in [1], is a modification to the standard DBSCAN algorithm. This adaptation enables the application of DBSCAN on data with a temporal dimension, such as data reflecting the movement of walking pedestrians over time.

2.1.1 Algorithm Description

The process involves running the original DBSCAN for each time step within the data. In the case of detecting groups from trajectories of walking pedestrians, the minimal size of clusters is 2. Therefore DBSCAN is run with *minPoints* = 2. The other parameter of DBSCAN, ***epsilon***, is for us to decide based on the specific data.

We monitor the number of time steps during which pedestrians share the same cluster. Ultimately, for each pair, we assess their total time steps (the union of distinct time steps of each pedestrian) and compare it with the instances they were together in a cluster. If this ratio exceeds the new parameter ***CTR (coexisting time ratio threshold)***, pedestrians are considered to belong to the same group.

Following pseudocode is describing Time-Sequence DBSCAN in closer detail:

ALGORITHM: TIME-SEQUENCE DBSCAN

Input: $T = \{T_1, T_2, \cdots, T_n\}$, epsilon, CTR

Output: $G = \{G_1, G_2, \cdots, G_m\}$

```
M ← 0^(n×n) // coexistence matrix, counts how many times (i,j) are in a cluster
for t in T: // t are all time steps that exists for any trajectory in T
    points ← [T_1[t], ···, T_n[t]]
    C ← DBSCAN(points, epsilon, 2) //minPoints = 2 for group detection
    for C_x in C:
        for each pair (i,j) in C_x: M[i,j] += 1
    end
end
cluster_pairs = []
for ∀ T_i, T_j in T:
    time ← union of time steps covered by a pair
    if M[i,j] / time > CTR:
        cluster_pairs ← cluster_pairs + pair
    end
end
G ← create clusters from cluster_pairs
return G
```

2.2 Agglomerative Hierarchical Clustering with Hausdorff Distance

Agglomerative clustering, or bottom-up clustering, is a traditional method for grouping observations that creates a cluster tree. Starting with individual observations, the tree progressively forms subclusters as one moves upwards. To prevent merging into a single large cluster, a stopping rule is needed for the agglomeration algorithm. Additionally, rules are necessary to determine which subcluster should merge next at each stage of the tree-building process [8].

Agglomerative Hierarchical Clustering with Hausdorff Distance, developed by Weina Ge, Robert T. Collins, and R. Barry Ruback [2], builds on pedestrian detection and multi-object tracking methods, drawing inspiration from sociological models of collective human behavior. It autonomously identifies small groups of individuals traveling together through a bottom-up hierarchical clustering approach. The innovation lies in utilizing a generalized, symmetric Hausdorff distance, specifically defined with respect to pairwise proximity and velocity. We will refer to Agglomerative Hierarchical Clustering with Hausdorff Distance as AHC.

2.2.1 Algorithm Description

For this algorithm, a different definition of trajectory is used. The trajectory is a set of tuples (*s,v,t),* where s is the position, *v* is the velocity vector, and *t* is the time. s_i^t is a position *s* of trajectory *i* at the time *t*.

The distance measure w_{ij} between two trajectories *i* and *j* is defined as:

$$w_{ij} = \frac{\sum_t w_{ij}^t}{\rho_{ij}|\Gamma|}, for\, i \neq j \wedge t \in \Gamma,$$

$$w_{ij}^t = \alpha N\left(\left\|s_i^t - s_j^t\right\|\right) + (1-\alpha) N\left(\left\|v_i^t - v_j^t\right\|\right),$$

$$\rho_{ij} = \sum_t \delta_t(i,j),$$

$$\delta_t(i,j) = \begin{cases} 1 & \left\|s_i^t - s_j^t\right\| < \Gamma_s \wedge \left\|v_i^t - v_j^t\right\| < \Gamma_v \\ 0 & otherwise \end{cases},$$

where *N* is a min-max normalization operator applied independently for each pair of trajectories to linearly scale their velocity and distance differences into the range [0, 1], and weight is defined as α = 0.7. Γ is a temporal overlap between trajectories *i* and *j*. Γ_s is a ***distance threshold***, Γ_v is a ***velocity threshold*** – both are inputs to the algorithm.

Modified Hausdorff distance introduced by Marie-Pierre Dubuisson and Anil E. Jain [9] is used to measure the inter-group closeness between two groups *A* and *B*:

$$H(A,B) = \frac{h(A,B) + h(B,A)}{2},$$

$$h(A,B)=\frac{\sum_{i=1}^{|A|}\sum_{j=1}^{\lceil\frac{|B|}{2}\rceil} d_{il}}{|A|\times\lceil\frac{|B|}{2}\rceil},$$

where d_{il} is the lth smallest distance amongst all the distances w_{ij} , j ∈ B.

AHC starts with each individual in their own cluster. During each iteration, we select groups A and B with the lowest distance $H(A,B)$ and try to merge them. If merge is accepted depends on ***intragroup tightness***.

For any group of size $k\geq 1$ we create graph G_k in which vertices represent members of the group. The edge between vertices n_i and n_j exists if $\rho_{ij}<\Gamma_t$. This represents a need for i and j to be "close together" for Γ_t time steps. The ***time coexistence threshold*** Γ_t is the last input needed for the algorithm.

Let e_k be the number of edges in G_k and $\hat{e}_{k+1}$ be the minimal number of edges desired in G_{k+1} after merging with a single person. By definition $e_1=\hat{e}_1=0$. A person can be added to the existing G_k if they are connected to at least half of the members of G_k:

$$\hat{e}_{k+1}=e_k+\lceil\frac{k}{2}\rceil.$$

We can derive: $\hat{e}_k=\left(\frac{k}{2}\right)^2$ for even *k,* and $\hat{e}_k=\frac{k-1}{2}\left(1+\frac{k-1}{2}\right)$ for odd *k.* Two groups G_p and G_q can be merged if they satisfy the ***intragroup tightness criterion***:

$$e_{p+q}\geq\hat{e}_{p+q}+(e_p-\hat{e}_p+e_q-\hat{e}_q).$$

If the groups don't satisfy the ***intragroup tightness criterion***, the merge is denied, and the algorithm continues with groups with the next lowest distance, etc. The algorithm stops and returns all groups after it fails to merge any two groups [2].

3 Human Behavior in Groups

This chapter delves into how people behave in groups, particularly when walking together, from a sociological and analytical standpoint. The analysis includes examining whether groups tend to cluster, form linear arrangements, or adopt more complex structures. Group size is also a key aspect we'll explore, investigating how and if it can provide useful insight into crowd dynamics.

Findings from this chapter will be a key for creating our own metric based on group shapes.

3.1 Group Shapes

In a study done by Mehdi Moussaid, Niriaska Perozo, Simon Garnier, Dirk Helbing, and Guy Theraulaz [10], it was observed that a significant proportion of pedestrians prefer walking in groups. In the study, two populations were studied, and in both, over half of pedestrians were walking in groups rather than alone – 55% and 70% of the population, respectively. It is also noted, that the environment influences this distribution, with leisure areas like commercial walkways showing a higher tendency for people to walk in groups.

When walking alone, a pedestrian needs to adapt its walking path to avoid other pedestrians and obstacles and to general infrastructure. In the case of an empty street, there is very little reason to observe anything other than a straightforward path from the starting point to the point of interest. This assumption changes when two or more people are walking together and talking to each other. In social groups, people adapt their walking paths to be able to talk to each other, gesture to each other, etc. This was observed in [4] [10].

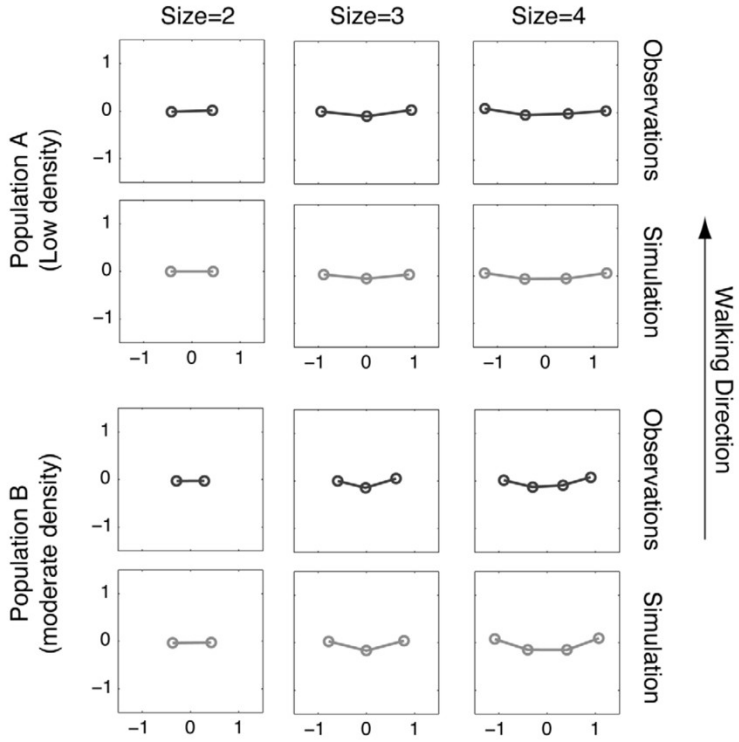

Figure 1: Group positions per crowd density and group size, Source: [9]

As seen in Figure 1 and Figure 2, people tend to walk next to each other from the view of the walking direction, forming a perpendicular line to the vector of the walking direction. This is easy to see in groups of two people but can also be observed in bigger groups. In groups of three, people usually walk in a wide, straight line, or in a "**V**"-shape, and in bigger groups, the shape could be described as a parabola.

Studies have shown that these shapes change with crowd density [10]. With increasing density, groups of two become closer to each other. Groups of three and higher are also closer to each other, and their "**V**"-shape gets more pronounced. This holds until the critical density is reached and the shape changes drastically. "In very high densities, V-like patterns are transformed into a lane aimed toward the direction of motion." [4]

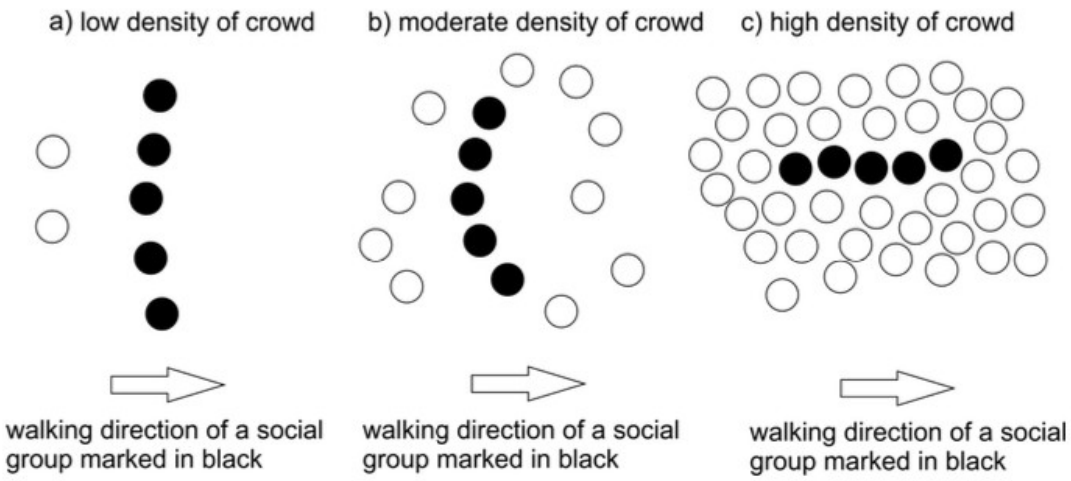

Figure 2: Group shapes per crowd density, Source: [4]

These group-shape findings will be one of the factors when we build a group detection algorithm.

3.2 Group Speeds

The study on the walking behavior of pedestrian social groups [10] found that the walking speed of pedestrians is influenced by both density levels and group size. Their observations can be seen in a Figure 3. At low density, individuals walk faster compared to higher density, aligning with prior research on pedestrian traffic [11]. Additionally, their observation reveals a linear decrease in pedestrian walking speeds with increased group size. Interestingly, the density level does not significantly impact the slope of this group-size-related speed decrease.

Another study was done at the Bus Terminal Area [12] using data collected through a 45-minute video recording at the main lobby of the bus terminal revealed that walking speed is dependent on multiple individual factors. Male pedestrians walked faster than female pedestrians, with average speeds of 1.13 m/s and 1.07 m/s, respectively. Additionally, pedestrians without baggage walked faster than those carrying baggage, indicating that baggage can impact walking speed due to additional weight and potential distractions.

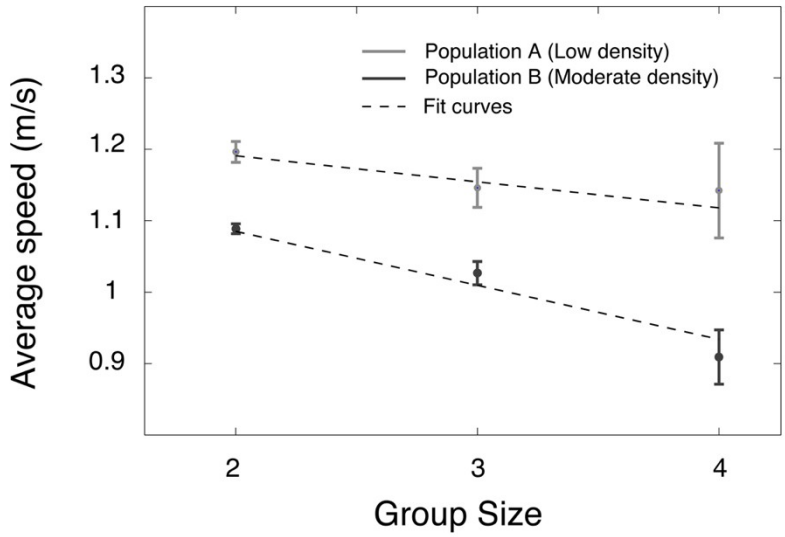

Figure 3: Average Speed per Group Size, Source [10]

According to [13], groups of people walking together exhibit common behavioral patterns, such as moving at the same speed and reforming quickly when separated [14]. Small groups often display unintentional synchronization in stepping, which can be a result of shared feelings of unity and closeness, a phenomenon known as "mirroring" in the psychology [15].

3.3 Group Sizes

Another dimension of group behavior to consider is the group size. As previously noted, the size of a group influences the walking speeds of pedestrians within that group. According to some studies, the most prevalent groups typically comprise only two to four members [10], making larger groups significantly less frequent.

4 Used Techniques

In this chapter, we will introduce all the techniques that we will need to build our algorithm. Firstly, we are going to explain how to deal with **misalignment of trajectories**. Next, we will introduce **shared time steps**, a measure that will quantify how many time steps two trajectories share. We will define the **distance** between two trajectories and the **cosine similarity** for two trajectories. At last, we will introduce **group shape score**, a new way how to quantify the expected shape of a group.

4.1 Trajectory Misalignment

When comparing the trajectories of two pedestrians, a significant challenge arises from the unavoidable discrepancy in trajectory lengths. This creates a problem of handling calculations that require values from both trajectories at a single time step, and only one trajectory has data for that time step.

We have opted for a straightforward approach: any measure requiring data from both trajectories will be computed exclusively on intervals during which data is available from both trajectories. This strategy presents the risk of yielding misleading results. For example, a person leaving the tracking area might exhibit a small average distance from another pedestrian entering the area from the same spot. We will define *common time steps* to use in this chapter.

Definition (Common Time Steps): Let $T_i = \left[\left(x_{i_0}, y_{i_0}\right), \cdots, \left(x_{i_n}, y_{i_n}\right)\right]$, $T_j = \left[\left(x_{j_0}, y_{j_0}\right), \cdots, \left(x_{j_m}, y_{j_m}\right)\right]$ be two trajectories, then t^{ij} will denote the intersection of their times steps:

$$t^{ij} = \{i_0, \cdots, i_n\} \cap \{j_0, \cdots, j_m\}.$$

This problem is not limited to the beginning or ends of trajectories but could also happen in the middle of tracking data with tracking errors or people leaving the area and returning. This is completely avoided by choosing to interpolate missing data – which we will describe in the chapter dedicated to the dataset.

4.2 Shared Time Steps

Since we are not penalizing other metrics for being calculated on potentially small amounts of data when two trajectories coexist, we must introduce a measure of shared time steps.

For any two trajectories T_i, T_j, let's define their shared time steps $St(T_i, T_j)$ as a percentage of time steps of the longer trajectory for which the shorter trajectory also exists.

Let $T_i = [(x_{i_0}, y_{i_0}), \cdots, (x_{i_n}, y_{i_n})], T_j = [(x_{j_0}, y_{j_0}), \cdots, (x_{j_m}, y_{j_m})]$ be two trajectories. Without loss of generality $|T_i| < |T_j|$. Then shared time steps will be calculated as:

$$St(T_i, T_j) = \frac{|t^{ij}|}{|T_j|}.$$

4.3 Distance

Distance is a core metric for group detection. For computing the distance between two trajectories, we decided to use the median of distances of their time-aligned positions. Unlike the mean distance, which is influenced by extreme values, the median is the middle observation, exactly at the center of the distribution. The median is a robust measure, meaning it is less sensitive to outliers or extreme values, unlike the mean.

When all distances are sorted, the median equals their middle point. For a list of distances of odd size *n,* the median equals the value on the *n/2*-th position. For even *n,* the median equals the mean value between the *n/2*-th distance and *(n/2)+1*-th distance. Let $T_i = [\boldsymbol{a}_{i_0}, \cdots, \boldsymbol{a}_{i_n}], T_j = [\boldsymbol{b}_{j_0}, \cdots, \boldsymbol{b}_{j_m}]$ be to trajectories, and D is a list of distances $\|\boldsymbol{a}_t - \boldsymbol{b}_t\|$ for every $t \in t^{ij}$, then their distance will be calculated as:

$$d(T_i, T_j) = med(D)$$

4.4 Cosine Similarity

Cosine similarity is a metric used to determine the cosine of the angle between two non-zero vectors in a multidimensional space. It is commonly employed in data analysis to assess the similarity between two vectors, regardless of their magnitudes.

The cosine similarity ranges from -1 to 1, where a value of 1 indicates perfect similarity, 0 denotes no similarity, and -1 implies perfect dissimilarity.

$$S_C(x, y) = \frac{x \cdot y}{\|x\| \cdot \|y\|},$$

where $x \cdot y$ is the dot product of vectors x and y, $\|x\|$, $\|y\|$ are lengths of vectors x and y, respectively.

For two trajectories T_i and T_j, and their ordered common time steps $t^{ij} = [0, 1, ..., n]$ we will define their cosine similarity as an average of the cosine similarity between their displacement vectors:

$$S_C(T_i, T_j) = \frac{1}{n-1} \sum_{t=1}^{n} S_c(T_i[t] - T_i[t-1], T_j[t] - T_j[t-1])$$

4.5 Group Shape Score

So far, all measures mentioned have been used strictly for pairs of trajectories. In this section, we will use knowledge gained from 3.1 to define the group shape score, a measure calculated for set of n trajectories.

Group shape score measures how the shape of the group corresponds to our expections.

4.5.1 Principal Component Analysis

Principal Components Analysis (PCA) is a method used for simplifying complex data structures, making it easier to understand relationships between variables. Unlike focusing on why events occur, PCA centers on providing insights into how different factors in a dataset interact. It serves the purpose of distillation rather than causal analysis [16]. PCA finds utility in various fields. In ecology, it helps interpret natural connections, while in food science, it untangles relationships among different food properties [16] [17].

The primary goals of PCA are to streamline complex data, aid in interpretation, and enhance the overall understanding of relationships within the dataset [18]. PCA's significance lies in its mathematical foundations. Through numerical evaluations, it simplifies datasets, facilitating subsequent analyses. PCA takes diverse measurements and condenses them into a few principal components, capturing

essential information and revealing the primary data structure [18]. Principal components are designed to encapsulate the maximum variance within a dataset. In mathematical terms, the first principal component is the line that maximizes the variance, ensuring the average squared distances from the projected points to the origin are optimized [19]. Each additional principal component shows in which direction there is the next greatest variability while orthogonal to each other.

We use PCA as a dimensionality reduction and data transformation technique to detect the shape of the group. At any given time, the first principal component of a group describes in what direction the group spreads the most – effectively describing its "shape." In isolation, this information does not mean anything. We also need to know the walking direction of the group – its displacement vectors. To make these we need to define groups position.

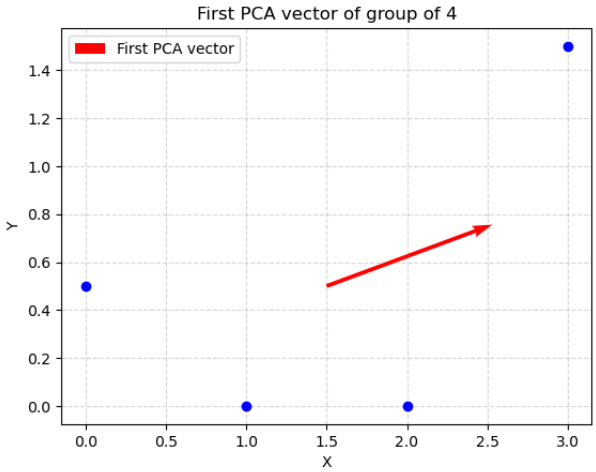

Figure 4: Example of the first PCA vector on four points

4.5.2 Centroid

We have positions for each trajectory in the group but not for the whole group. We will define the group's position as its centroid. The centroid of n 2D points is the point that represents the geometric center or average position of the set of points. It is a point that balances the distribution of points in both the x and y dimensions. The centroid C of a group G with n trajectories at time step t can be calculated by taking the mean of the positions of all trajectories included in the group G.

$$C_t(G) = \frac{1}{n} \sum_{i=0}^{n} T_i[t]$$

4.5.3 Evaluation

Now that we have a displacement vector and a vector that describes the group's shape, we just need to compare them.

If a group has a perfectly "wide" shape, its shape vector is perpendicular to its displacement vector. On the other hand, a group that has a perfectly "long" shape has its shape vector and velocity vector parallel to each other. Visualization of this can be seen in Figure 5.

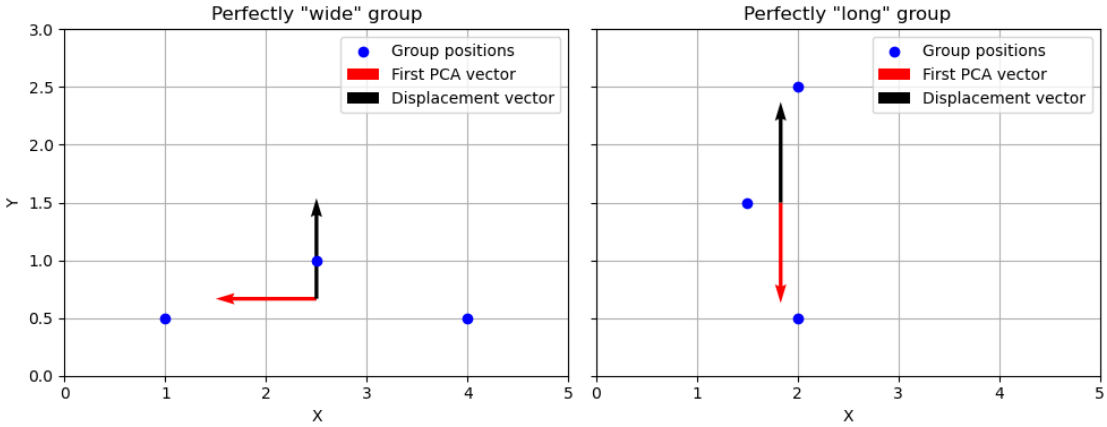

Figure 5: Example of "wide" and "long" group

We need to define a function that would return the highest value for perpendicular vectors and the lowest for parallel vectors. We define the group shape score of group G at time t as:

$$Gs_t(G) = \begin{cases} \dfrac{\theta_t}{90} & if\ 0 \le \theta_t \le 90 \\ 1 - \dfrac{\theta_t - 90}{90} & if\ 90 < \theta_t \le 180 \end{cases}$$

where θ_t is the angle between the velocity vector and the first principal component vector of a group at time t. θ_t is in degrees and modulo 180. This function will linearly map angles from 0 to 90 degrees to interval [0,1] and between 90 and 180 degrees to interval [1,0]. $Gs_t(G) = 1$, for perfectly "wide"-shaped group G, and $Gs_t(G) = 0$, for perfectly "long"-shaped group G.

The total shape score of the group G is the average shape score across time steps that the whole group exists at:

$$Gs(G) = \frac{1}{|t^G|} \sum_{t \in t^G} G s_t(G),$$

where t^G is the intersection of all time steps of all trajectories in the group.

4.6 Breadth-First Search

Breadth-First Search (BFS) is a fundamental graph exploration algorithm, defined for a graph *G(V, E)* with vertices *V* and edges *E*. It starts from a specified source vertex *s*, systematically traversing the edges to discover every reachable vertex from *s*. The algorithm employs a first-in, first-out (FIFO) queue to manage the exploration order, progressing in waves to uncover vertices at increasing distances from the source [20].

The BFS will be used in our algorithm to discover a connected component in a graph in which trajectories represent vertices and edges between them represent their 'similarity'.

4.7 Hyperparameter Optimization

Hyperparameter optimization is the method of adjusting settings in an algorithm to maximize its performance. This involves systematically exploring different values for these settings, aiming to minimize errors or enhance performance on validation data [21]. Different methods can be used to achieve this, for example, Grid search, Random search, Bayesian Optimization, or Evolutionary algorithms [22]. In this thesis, we will focus solely on Evolutionary algorithms to approximate optimal parameters for our algorithm.

4.7.1 Evolutionary Algorithms

Evolutionary Algorithms (EAs) operate on a population of potential solutions, employing the survival of the fittest principle to sequentially improve approximations to a solution. In each generation, individuals are selected based on fitness function and undergo reproduction with variation operators. Evolutionary Algorithms encompass various techniques, such as evolutionary programming, genetic programming, evolutionary strategies, or swarm intelligence. EAs are inspired by biology, using crossover, mutation, and natural selection mechanisms to progressively improve potential solutions. The typical EA begins by initializing a population, evaluating fitness of individuals, generating a new generation through breeding, and combining parent and child solutions to find optimal solutions [23].

Fitness refers to the measure of how well an individual solution in the population performs. Crossover entails combination of genetic information from two parent individuals to produce offspring. Mutation introduces random modifications to an individual's genetic information. Individuals exhibiting higher fitness are favored by natural selection, increasing their chances of being selected for reproduction [23] [24].

Our primary focus in the context of this thesis will be on an evolutionary optimization.

4.7.2 Evolutionary Optimization

Evolutionary optimization involves applying an EA to solve problem of hyperparameter optimization. The process starts with a population of randomly generated hyperparameter configurations, their performance is evaluated by fitness function and individuals are ranked accordingly. New members replace the worst performers with hyperparameter values derived through mutation or recombination of the best performers. This process continues until a stopping condition is met, evolving the population toward an optimal solution [22]. The method is visualized in Figure 6.

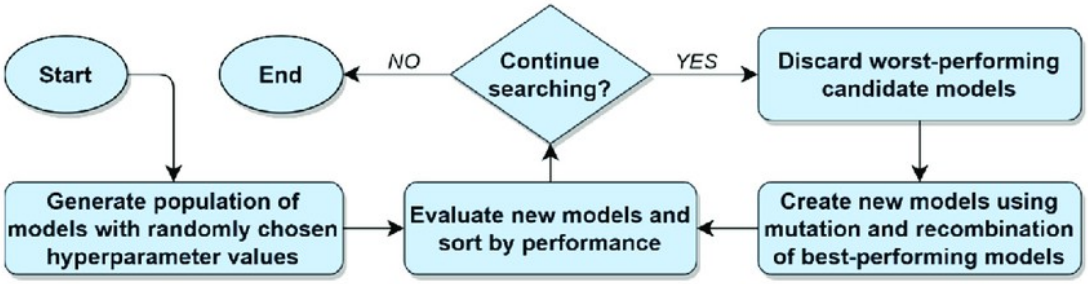

Figure 6: Hyperparameter optimization using the evolutionary method; Source [22]

4.8 k-fold Cross-Validation

Cross-validation is a statistical method for the comparison and evaluation of algorithms. It works by dividing data into two segments: one for training a given algorithm and the second for validating the trained model. In the standard cross-validation procedure, training and validation sets alternate in successive rounds, ensuring that each data point is validated. [25]

Cross-validation is widely used in machine learning, data mining, and similar areas for performance optimization, offering a systematic approach to help train prediction models.

K-fold cross-validation is a type of cross-validation where data is partitioned into k equally sized segments. Through k iterations of training and validation, each round uses a k-th fold for validation and the remaining folds (k-1) for training. This approach enables thorough and comprehensive evaluation [25] [26]. The process is described in Figure 7.

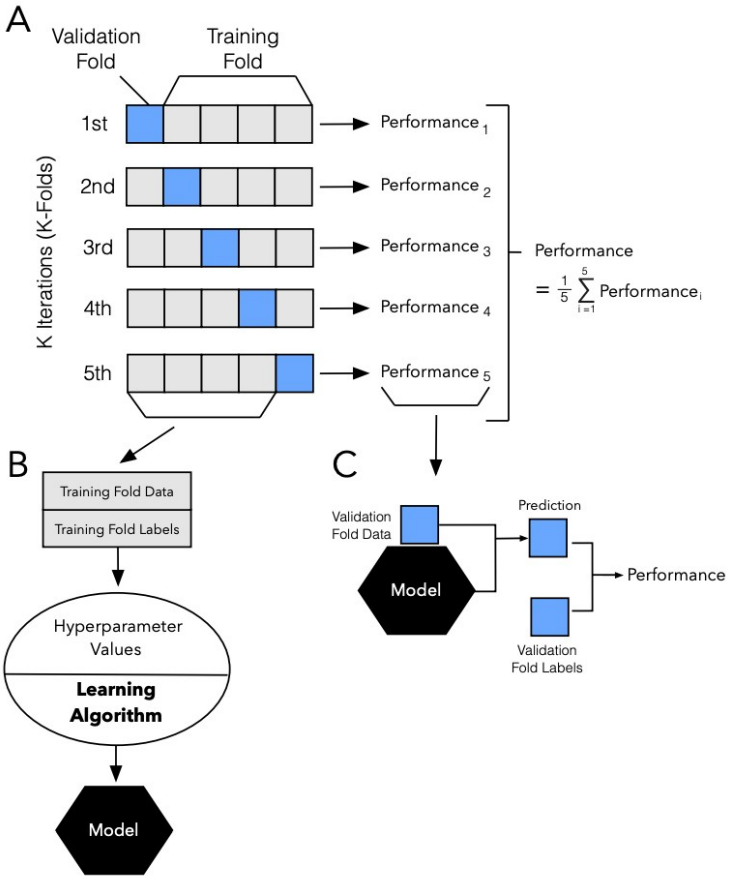

Figure 7: Illustration of k-fold cross-validation, Source: [26]

5 Proposed Solution

In this chapter, we will show an algorithm composed of techniques and ideas presented in previous chapters. Our algorithm consists of two main steps for each pedestrian. The first is to generate group candidates based on being "similar" enough. To decide which trajectories are "similar" we will use distance, cosine similarity, and shared time steps. The second part is creating a group out of the candidates based on group shape scores using a greedy algorithm.

5.1 Algorithm Description

At the beginning of the algorithm, we calculate three similarity measures for each pair of pedestrians:

1. distance similarity – measured by $d(T_i, T_j)$ from 4.3,
2. cosine similarity – measured by $Sc(T_i, T_j)$from 4.4,
3. shared time steps – measured by $St(T_i, T_j)$ from 4.2.

These measures will define how 'similar' two trajectories are. For every measure, there exists a corresponding threshold in the algorithm's input – $thr_m = [thr_d, thr_{sc}, thr_{st}]$, that two trajectories have to pass to be considered 'similar'. We create graph S in the following way:

- for each $T_i \in T$, create node n_i,
- edge e_{ij} between n_i and n_j exists if all conditions are met:
    1. $d(T_i, T_j) < thr_d$,
    2. $Sc(T_i, T_j) > thr_{sc}$,
    3. $St(T_i, T_j) > thr_{st}$.

ALGORITHM: OUR GROUP DETECTION ALGORITHM

Input: trajectories $T=\{T_1, T_2, \cdots, T_n\}$, measure thresholds thr_m, groups shape score thresholds thr_{gs}

Output: detected groups $G=\{G_1, G_2, \cdots, G_m\}$

```
1:  S ← getMeasureThresholdSatisfactoryGraph(T, thr_m)
2:  G ← {} //empty set
3:  j ← 1 //group counter
4:  P ← T //pool
5:  for each T_i in P
6:      C_i ← run BFS(T_i)
7:      G_j ← getGroupUsingGroupShapeScore(T_i, C_i, thr_gs)
8:      G.add(G_j)
9:      P.remove(G_j) // removes each trajectory in G_j from the pool
10     j++ //increase group counter
11: end
12: return detected_groups
```

The pool of all pedestrians is created, and then we start looping through the pool. From the pool, we randomly select a trajectory T_i. We are interested in finding pedestrians who could create a group with T_i (group candidates for T_i). We identify group candidates for T_i by conducting a BFS on S starting from a node n_i. The result of BFS is the connected component C_i. By definition of edges in S, this component is composed of nodes that are 'similar' to trajectory T_i either directly (neighbors), or indirectly – for nodes that are not connected to n_i, but there exists a path from n_i to them.

Once we have C_i, we move to the second part of the algorithm – selecting those pedestrians that will create a group with the T_i based on the group shape score. For next steps, the graph S is irrelevant – was only used to get candidates.

In the beginning, the group consists only of the T_i. We check each candidate to see if combining it with the current group would create a new valid group. For each group size, we have defined the group shape score threshold (input to the algorithm –

$thr_{gs}=[thr_2, thr_3, thr_4, thr_{5+}]$) the group needs to pass to be valid. For group sizes of 5 or bigger, a single threshold is used. For example, thr_{gs} = [0.4, 0.3, 0.1, 0.05] means that a group size of two has to have a shape score of 0.4 or higher, 0.3 for a group of three, 0.1 for a group of four, and 0.05 for a group of five and bigger.

Once we find such a candidate, we immediately add it to the current group and repeat the process – even with candidates that were previously declined. Some candidates might be declined and still form a valid group when more candidates are added.

METHOD: GET GROUP USING GROUP SHAPE SCORE

Input: trajectory T_i, group candidates C_i, groups shape score thresholds thr_{gs}

Output: group G_j

1: $G_j \leftarrow \{T_i\}$

2: C_i.remove(T_i)

3: $T_{new} \leftarrow$ getShapeFulfillingTrajectory(G_j, C_i, thr_{gs})

4: **while** T_{new} **is not** None:

5: $\quad G_j$.add(T_{new})

6: $\quad C_i$.remove(T_{new})

7: $\quad T_{new} \leftarrow$ getShapeFulfillingTrajectory(G_j, C_i, thr_{gs})

8: **end**

9: **return** G_j

Since this is a greedy approach, it is possible that we add a candidate to the group that prevents us from discovering a better group. This is an accepted risk since we cannot test all possible subsets of all candidates – for *n* candidates we would have to test 2^n groups for their group shape score.

We stop once going through all the remaining candidates does not yield a new group member. It is possible that no candidate is added to the group and the only member is the T_i. The group is added to the list of groups, all its members are removed from the pool, and the process continues with another pedestrian from the pool.

METHOD: GET SHAPE FULFILLING TRAJECTORY

Input: current group G_j, group candidates C_i, groups shape score thresholds thr_{gs}

Output: trajectory T_{new}

1: size $\leftarrow |G_j|+1$ //size $\leftarrow$ 5 if size > 5

2: **for each** T_x **in** C_i**:**

3: $\quad G_j^* \leftarrow G_j \cup T_x$

4: $\quad$ **if** $Gs(G_j^*) > thr_{gs}[\text{size}]$: **return** T_x

5: **end**

6: **return** None

6 Dataset

There are many available datasets of tracking data for pedestrians in public spaces, but not many include ground truth data for social grouping. In this chapter, we will introduce the dataset used for this thesis, what preprocessing was needed, and general statistics.

6.1 DIAMOR Dataset

Using a laser range-finder tracking system, researchers in Japan tracked the movement of pedestrians on two separate days in "two large straight corridors connecting the Diamor shopping center in Osaka, with the railway station" [3]. For social group annotations, video cameras were used, and from recorded video, two members of nontechnical staff were asked to label groups [27]. The DIAMOR dataset is available at [3].

Additionally, tracking data for 6 days from the Asia and Pacific Trade Center (ATC) in Osaka is available. For these, only a specific part of the area was covered with video cameras, and groups that did not move through this area were not labeled. This limitation is problematic for us, and therefore we did not use this data.

6.2 Data Format

For both days, two files are available – a personal tracking file and a group file. A person tracking file is a comma-separated file where each row represents data for one tracked person at a fixed time. Columns are:

- time [s],
- pedestrian_id,
- position_x [mm],
- position_y [mm],
- velocity [mm/s],
- angle_of_motion (direction of velocity vector) [rad],
- facing_angle (body direction) [rad].

We processed this file and removed facing_angle because even if this parameter could improve group detection, it is not a common variable in datasets and its inclusion would make any findings less useful for datasets without it. Also, instead of velocity and angle_of_motion, we calculated velocity_x and velocity_y after interpolating and downsampling the dataset, which will be described later.

The group file is a text file where each line represents one pedestrian (only those in any group are included), and it contains the following space-separated fields:

- pedestrian_id,
- group_size,
- partner_id_1,
- list of IDs of all other pedestrians in the group,
- number_of_interacting_partners,
- interaction_partner_id_1,
- list of IDs of all socially interacting partners

Interaction partners were removed for the same reason as facing_angle from the tracking file. We worked with groups as sets of pedestrian IDs.

6.3 Removal of Unsuitable Pedestrians

The first step that needed to be done was the removal of trajectories that were too short or too long. Neither extreme is interesting for our task – a trajectory that is too short does not have enough data to properly observe, and a trajectory that is too long belongs to a person who is probably walking nonstandardly. We removed any pedestrian with a trajectory shorter than 8 seconds or longer than 120 seconds.

For the first day, we removed around 25% of all pedestrians – only 37 for having a long trajectory. Unfortunately, it seems that the data for the second day are not optimal, and we had to remove more than half of the pedestrians due to their short trajectories. Even if we decided to lower the threshold to 4 seconds, we would still have to remove 12639 pedestrians. These unnaturally short trajectories raise a question about the data quality for the second day.

For some groups, not all pedestrians were properly tracked – in such cases, a pedestrian's ID is replaced with "-1". Our algorithm uses different group shape score thresholds for different group sizes. These groups might be problematic as their group shape score will be compared to an incorrect threshold. Therefore, we decided

to remove groups with untracked pedestrians completely from our data – including trajectories of all pedestrians tracked properly in that group. Additionally, some special cases were also removed from the dataset. For example, a parent with a baby stroller was labeled as a group of two people, which does not align with our definition of a group, and it is of no interest to us. For all these reasons, we had to remove 52 pedestrians from day 1 and 92 pedestrians from day 2. At the end there are 7346 pedestrians for day 1 and 11907 for day 2. The summary of statistics is in Table 1.

	Day 1	Day 2
Original number of trajectories	9969	27639
Removed (short)	2534	15579
Removed (long)	37	61
Removed (other reasons)	52	92
New number of trajectories	**7346**	**11907**

Table 1: Statistics for trajectories in DIAMOR dataset

6.4 Downsampling of Trajectories

In [12], it is stated that pedestrians walk at a speed of ~1m/s. On average, this dataset has a gap between two consecutive time steps of ~0.042s or ~24 updates per second. That means the average pedestrian moves by a few centimeters between two time steps. This level of detail is unnecessary for our use case and will only slow down any algorithm used. We downsampled the data, aiming for 0.25s between each time step.

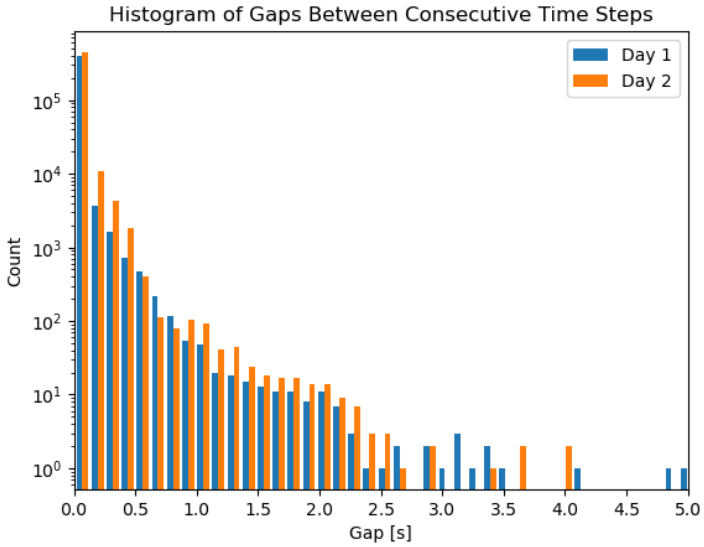

Figure 8: Histogram of time gaps between consecutive time steps in DIAMOR dataset

Unlike video cameras that record with fixed frames per second, this dataset has irregular gaps between consecutive time steps. We cannot have a constant gap unless we interpolate the majority of data. At no point is that necessary for us, and we would rather interpolate less. Therefore, we iterated through each time step, accumulating a time gap until the cumulative gap exceeded a quarter of a second. Subsequently, we marked the last time step or the one immediately before it, selecting the one that was closer to the target gap of a quarter of a second. Reset the counter and repeat until the last frame. We kept only data associated with marked time steps and removed every other from the dataset.

6.5 Interpolation of Missing Data

Observing the histogram of time gaps between consecutive time steps in Figure 8, it is evident that interpolation is also necessary to ensure a desired gap of a quarter of a second. We interpolated data if the gap between two time steps was over half a second. For any gap like that, we insert a new time step in the middle of a gap. For larger gaps, we inserted more time steps proportionally.

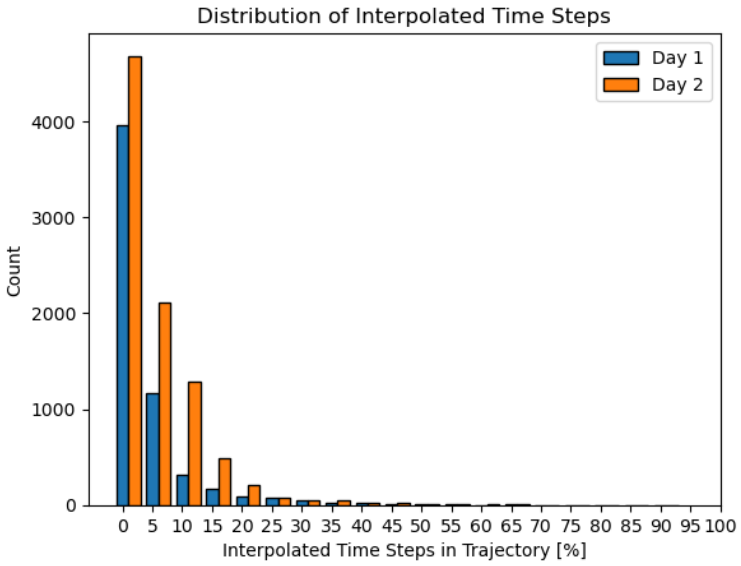

Figure 9: Distribution of interpolated time steps

On day 1, 87% of pedestrians have less than 10% time steps interpolated and 78% on day 2. Distribution of interpolated time steps is visible in Figure 9.

6.6 Descriptive Statistics

As mentioned, after removing unsuitable pedestrians, there are 7346 and 11907 pedestrians for day 1 and day 2, respectively. These pedestrians formed 728 groups on day 1, and despite an increase of 62% in pedestrian count on day 2, the increase in the number of groups was marginal, amounting to two additional groups. The number of groups per size can be seen in Table 2.

Group Size	2	3	4	5+	All
Day 1	627	73	21	7	728
Day 2	630	82	12	6	730

Table 2: Number of groups per size in DIAMOR dataset

6.6.1 Speed of pedestrians

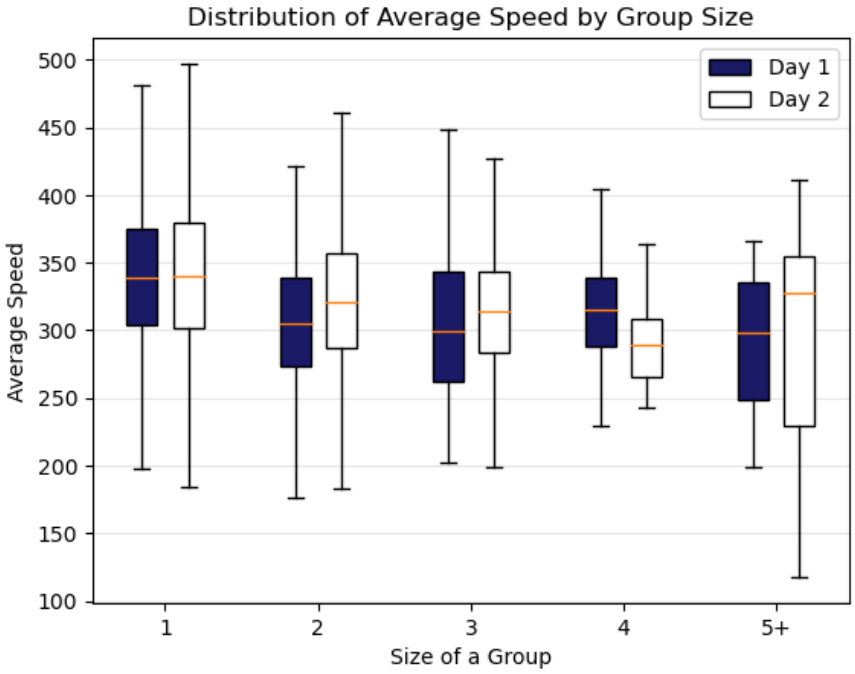

Figure 10: Distribution of average speed by group size

The average speed of pedestrians is illustrated in Figure 10. Due to the similarities of average speeds per group size, we are probably not able to use it for group size categorization.

6.6.2 Density

On average, there are 12 pedestrians at each time step on day 1 and 20 on day 2, with standard deviations of 4.7 and 7.4, respectively (Figure 11).

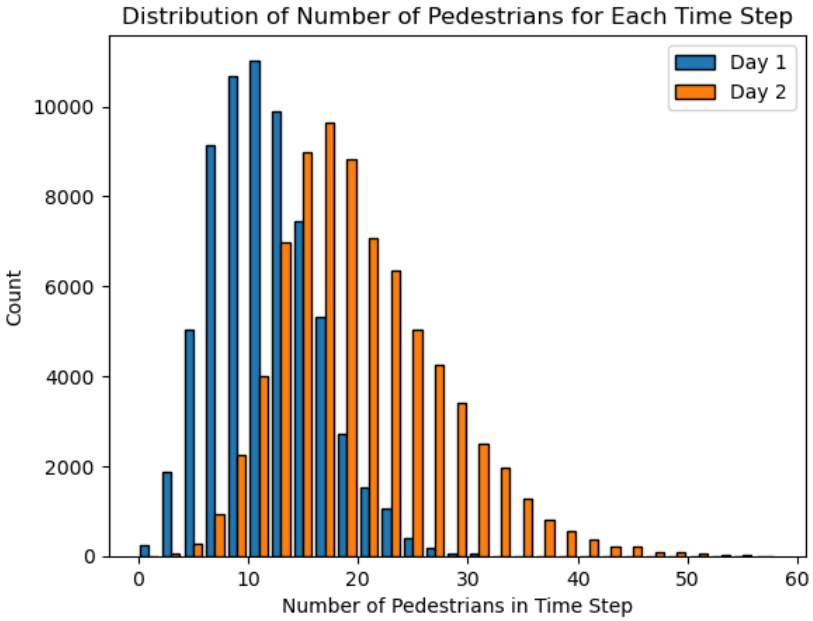

Figure 11: Distribution of pedestrians in time steps

As described in 3.1, one of the key factors influencing the behavior of pedestrians and the group shapes they form during locomotion is the proximal density of surrounding pedestrians. For lower densities, groups form "wide" shapes in the direction of their locomotion, and only when the density reaches a critical point the "long" shapes are preferred.

In a study [28], where the highest observed pedestrian density reached 4.4 pedestrians per square meter, researchers identified creations of river-like or line formation among pedestrians. However, they acknowledged the difficulty in detecting such situations and emphasized the need for further investigation. The study highlighted that in these instances of high density, the formation of lines could potentially be attributed to individuals creating obstacles, prompting others to organize themselves into a more compact shape for efficiency.

Regrettably, even if we account for removed pedestrians from the dataset, on neither day 1 nor 2, densities aren't high enough to properly test "long" shapes of groups. **Consequently, our algorithm will be constrained to assume that long shapes are never anticipated**.

6.6.3 Group Shape Score

For each group from the ground truth provided, we measured the group shape score as described in 4.5 to see if the findings from 3.1 are valid for our dataset. As mentioned above, we only expect wide shapes with group shape scores above 0.5. Generally, a trend of decreasing scores with increasing group size is observed on Figure 12. Median values for day 1 are 0.82, 0.71, 0.33, and 0.48 for group sizes of 2, 3, 4, and 5+, respectively. For day 2, median values are 0.85, 0.81, 0.57, and 0.28 for the same sizes.

For sizes two and three groups are forming mostly wide shapes, as was anticipated. Especially for groups of two pedestrians, we have more than 80% and 85% of groups with a group shape score higher than 0.7 on day 1 and day 2, respectively.

It is important to keep in mind that for groups of size four, we have only 21 groups on day 1 and 12 groups on day 2. For groups of size five and bigger we have less than 10 groups for both days. That is not a big enough sample size to make any conclusions, but it needs to be noted that the values are lower than we expected and our algorithm might struggle for groups of these sizes. One more notable observation is that the biggest groups on day 2 not only were, in general, not wide, but even their maximum group shape score was 0.45. Meaning no group from that category could be described as “wide”. It is possible that more research in big groups is needed.

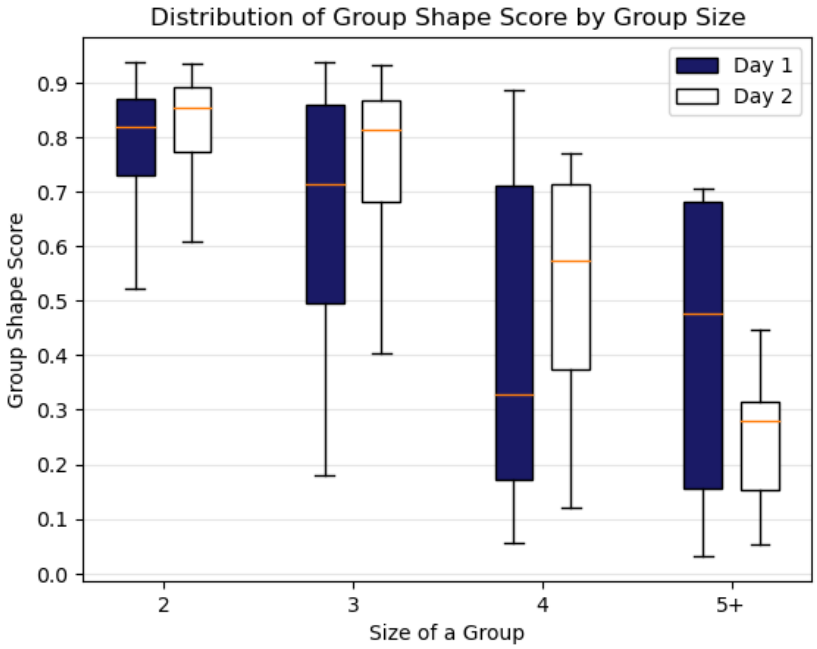

Figure 12: Distribution of group shape scores per group sizes

7 Experiments

In this chapter, we perform experiments on our algorithm and previously introduced existing solutions for the group detection problem. We properly tested all algorithms only on day 1 of the DIAMOR dataset. We identified an anomaly on the day 2 part of the dataset, which we will explore in section 7.7.

7.1 Evolutionary Optimization

Our proposed algorithm contains many different parameters that need to be decided on before we can test it. These parameters include a distance threshold, a cosine similarity threshold, a shared time threshold, and, for each group size, a group score threshold. A straightforward way to optimize the solution would be to experimentally test all combinations of values and decide on the best-performing ones. Regrettably, this method is time-inefficient due to the vast number of possible values. Additionally, each parameter is interdependent on others, resulting in a complex, multidimensional search space that is challenging to navigate through. To address this problem, we decided to utilize a hyper-parameter evolutionary algorithm, as mentioned in 4.7, to approximate the best possible values.

7.1.1 Hyper-parameter Optimization for Our Algorithm

We define individuals for the evolutionary algorithm as a vector of parameters of our algorithm from 5.1 that we wish to optimize:

1. **Distance threshold** (thr_d) – a number larger than 0,
2. **Cosine similarity threshold** (thr_{sc}) – a number between -1 and 1,
3. **Time shared threshold** (thr_{st}) – a number between 0 and 1,
4. **List of group shape score thresholds** (thr_{gs}) – numbers between 0 and 1.

The values in thr_{gs} are always sorted from the highest to lowest (do not increase with size). From the analysis in 6.6.3, this is a valid limitation to use.

To calculate the fitness function for an individual, we run our algorithm with parameters held by an individual. This returns a list of detected groups, which we can

compare to the ground truth group list by calculating the ARI value 1.4. This ARI value is the fitness of an individual. The higher the fitness value of an individual, the better the solution is.

Our algorithm uses **roulette wheel selection** [29]. Using this selection the probability of being selected is proportional to the fitness of each individual, mimicking the concept of a roulette wheel in a casino. The probability p_i of picking individual i is:

$$p_i = \frac{fitness_i}{\sum_{j=0}^{n} fitness_j},$$

where n is the number of all individuals.

We used a **uniform crossover** [30]. For each variable in an individual, we produce a random boolean – each represents the parent from which the variable should be taken for the first offspring. The second offspring is built by variables not chosen by the first offspring. Please remember that shape score thresholds are considered to be one variable. Example in Table 3.

	distance	cosine similarity	shared time	shape score list
random boolean	0	1	1	0
parent 0	1300	0.4	0.5	[0.5, 0.4, 0.3, 0.2]
parent 1	2000	0.2	0.75	[0.9, 0.8, 0.7, 0.6]
offspring 0	1300	0.2	0.75	[0.5, 0.4, 0.3, 0.2]
offspring 1	2000	0.4	0.5	[0.9, 0.8, 0.7, 0.6]

Table 3: Example of uniform crossover

For mutation, we chose to use the **Gaussian mutation** [31]. The Gaussian mutation for each variable separately adds a random value generated from a Gaussian distribution. We mutate every new offspring generated from the crossover method, but each variable has a separate **mutation rate** – the probability that the variable will be changed. Note that this is the only place where **shape score thresholds behave** as separate variables – each value has its own mutation rate.

The final hyper-parameter searching evolutionary algorithm will look as follows:

ALGORITHM: HYPER-PARAMETER SEARCHING EVOLUTIONARY ALGORITHM

Input: size of population, number of generations, parameters for mutation

Output: Individual with best fitness

```
population ← init() //initialize 0-th population with random individuals
calculate fitness for each in population
for i = 0 to number of generations:
    parents ← population
    new_offspring ← []
    while parents not empty:
        parent_0, parent_1 ← randomly pop two individuals from parents
        offspring ← crossover(parent_0, parent_1)  //produces two
        new_offspring ← new_offspring + offspring
    end
    for individual in new_offspring:
        mutate(individual)
        fittness(individual)
    end
    population ← select(population, new_offspring)
end
return max(population) //best individual from last population
```

We run EA with these parameters for all experiments on our dataset:

- population = 40,
- number of generations = 60,
- mutation rates for:
    - distance threshold = 0.15,
    - cosine similarity threshold = 0.15,
    - shared time threshold = 0.1
    - each value in the group shape score threshold = 0.1.

7.2 Validation

To validate the results, we use k-fold Cross-Validation with k = 4. This means we perform four iterations (folds). Each iteration involves using 75% of the pedestrians in the dataset for training purposes, and the remaining 25% is reserved for testing the optimal model identified during the training phase. The order of pedestrians is based on the time of their first appearance in the dataset. For the first iteration, the first quarter of pedestrians is used as a testing set. For the second iteration, the second quarter, and so on. It is a common practice to shuffle the data as a pre-processing step, but in the context of our task and dataset, shuffling is not appropriate due to linear nature of the data. Results will be added and averaged to calculate the total success rate.

7.3 Performance of Our Algorithm

The best results for each fold were as follows:

- For k = 1, thr_d = 1320, thr_{sc} = 0.31, thr_{st} = 0.29, thr_{gs} = [0.34, 0.15, 0.06, 0], ARI = **0.925**.
- For k = 2, thr_d = 1320, thr_{sc} = 0.21, thr_{st} = 0.29, thr_{gs} = [0.34, 0.29, 0.18, 0.04], ARI = **0.837**.
- For k = 3, thr_d = 1280, thr_{sc} = 0.16, thr_{st} = 0.26, thr_{gs} = [0.35, 0.27, 0.02, 0], ARI = **0.894**.
- For k = 4, thr_d = 1322, thr_{sc} = 0.28, thr_{st} = 0.29, thr_{gs} = [0.34, 0.15, 0.06, 0.04], ARI = **0.919**.

The average across all ARI results of 0.925, 0.837, 0.894, and 0.919 is **0.894**. The results for all folds appear to be consistent, except for the second fold, with a score lower by ~0.06 than the average.

To interpret the results, we combined all four test results together, and in the following two sections, we will analyze the data.

7.3.1 Correctly Detected Groups by Group Size

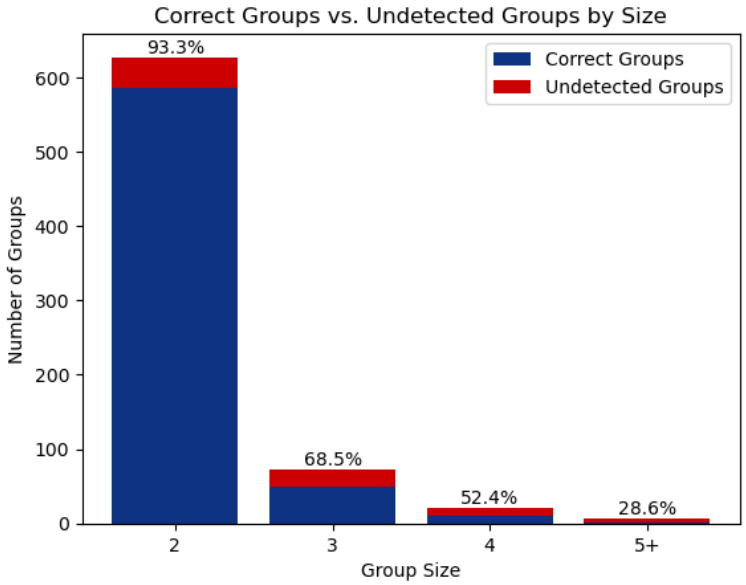

Figure 13: Distribution of detected ground truth groups for our algorithm

Group size	2	3	4	5+	All
Number of groups	627	73	21	7	728
Detected	585	50	11	2	648

Table 4: Number of detected ground truth groups for our algorithm

Table 4 and Figure 13 illustrates a percentage of successfully detected groups compared to ground truth data categorized by group size. We can observe a decrease in success rate with the increase in group size – from 93.3% for groups of size two down to 28.6% for groups of size five and bigger.

For groups of two, we have a high success rate and a large sample size – detecting 585 groups and failing to detect only 42. Groups consisting only of two pedestrians have the easiest dynamics, therefore they are more easily predicted. In larger groups, the predictability could be problematic. Individuals in bigger groups can momentarily pause, disengage from the group, or alter their trajectory, thereby inducing changes to the overall group shape score.

Due to the small number of groups of size five or bigger, it is not possible to conclude with a high degree of certainty whether our algorithm is ineffective at detecting bigger groups or if these results are influenced by chance. However, considering the decrease in performance even for groups of three, it is likely that our algorithm is not well suited for the detection of large groups.

7.3.2 False Positives by Group Size

We can gain interesting insights when we take a look at the number of false positives detected by our algorithm. We differentiate between a fully false positive group and a partially correct one. A group G can be partially correct for two reasons. Either $G \subset G_t$ for some ground truth group G_t. Or the other way around, $G_t \subset G$. This means, that either algorithm omitted some pedestrians in the detected group or added some that do not belong to the group.

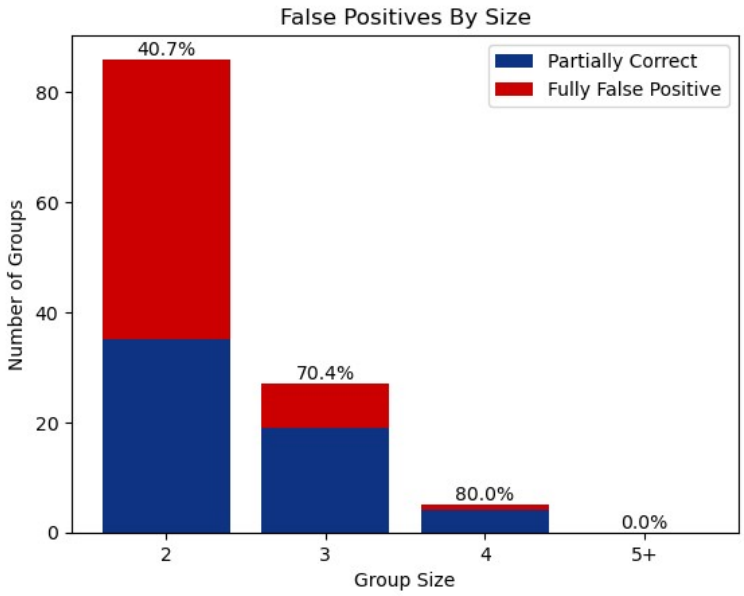

Figure 14: Distribution of false positives per group size for our algorithm

Group size	2	3	4	5+	All
Number of false positives	86	27	5	0	118

Table 5: Number of false positives for our algorithm

As seen in Table 5 and Figure 14, the highest number of false positives are in the first category for the groups of size two – 86 groups were incorrectly detected and 40% of them are partially correct. Out of 27 false positives for groups of size three, 9 of them are groups of size four or bigger that are missing one or more pedestrians. Since our algorithm failed to detect 15 groups of size four or bigger, this implies that 60% were detected as groups of size three.

7.4 Performance of Time–sequence DBSCAN

Time-sequence DBSCAN is easier to test as it requires only two parameters – epsilon and coexisting time ratio threshold (CTR). We used the same 4-fold cross-validation as with our algorithm. In this case, due to the small number of parameters, we didn't use an evolutionary algorithm, but we chose to search the parameter space for the correct configuration. We limited ourselves to a search space of epsilon values ranging from 1100 to 1500, incrementally examined at intervals of 50. From our additional testing, it was clear that epsilon below 1100 and above 1500 was not optimal. The dataset's positions of pedestrians are in millimeters, so a change of 50 equals a change of 5 cm. CTR ranges from 0 to 1, and we tested it from 0.2 to 0.95, with intervals of 0.05.

The best results for each fold were as follows:

- For k = 1, epsilon = 1250, CTR = 0.4, ARI = **0.899**,
- For k = 2, epsilon = 1200, CTR = 0.3, ARI = **0.835**,
- For k = 3, epsilon = 1450, CTR = 0.6, ARI = **0.890**,
- For k = 4, epsilon = 1250, CTR = 0.4, ARI = **0.889**.

The average across all ARI results is **0.878**. The performance is only marginally worse than that of our algorithm. Again, we can see consistent results, with the only deviation being the second fold, the result of which is way lower than the others.

7.4.1 Results by Group Size

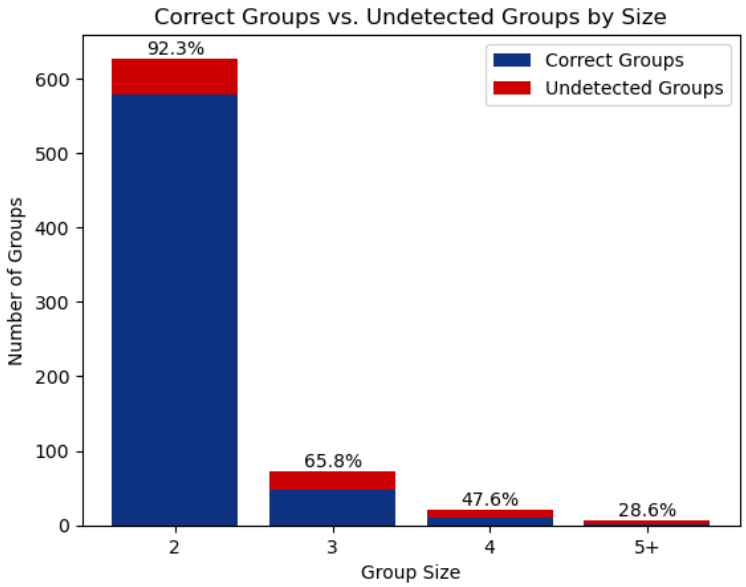

Figure 15: Distribution of ground truth groups for Time-Sequence DBSCAN

Group size	2	3	4	5+	All
Number of groups	627	73	21	7	728
Detected	579	48	10	2	639

Table 6: Number of detected ground truth groups for Time-Sequence DBSCAN

Time-sequence DBSCAN's performance is very similar to that of our algorithm, even when looking at the success rate per group's size (Figure 15 and Table 6).

The number of detected groups of size two is almost the same. Our algorithm was able to detect 93.3% of groups of size two versus 92.3% detected by Time-sequence DBSCAN – a difference of one percentage point equals to 6 more groups. Groups of sizes three and four have bigger differences but looking at the absolute numbers, our algorithm detected two more groups of size three and one more group of size 4. The result for the groups of the biggest size is the same.

7.4.2 Incorrectly Detected Groups by Group Size

Compared to our algorithm, Time-sequence DBSCAN detected 19 more false positive groups. The partially correct percentages are around the same levels. Unlike our algorithm, we also see false positives in the biggest category, and all three of them are partially correct (Figure 16 and Table 7).

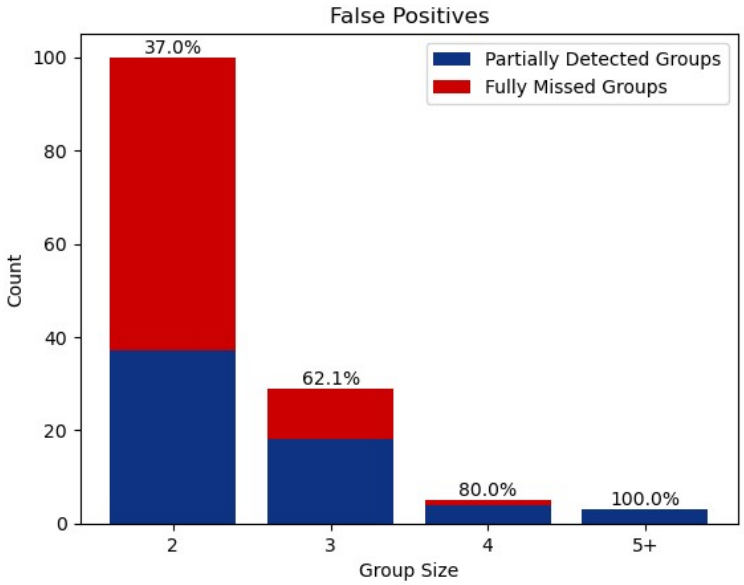

Figure 16: Distribution of false positives per group size for Time-Sequence DBSCAN

Group size	2	3	4	5+	All
Number of false positives	100	29	5	3	137

Table 7: Number of false positives for Time-Sequence DBSCAN

7.5 Performance Agglomerative Hierarchical Clustering

For testing of AHC we used the same technique as for Time-sequence DBSCAN. Distance threshold Γ_d ranged from 1000 to 1600 in intervals of 100. Velocity threshold Γ_v ranged from 80 to 170 in intervals of 10. Time coexistence ratio threshold Γ_t ranged from 16 to 32 in intervals of 4. The best results for each fold were as follows:

- For k = 1, $\Gamma_d = 1300$, $\Gamma_v = 130$, $\Gamma_t = 20$, ARI = **0.856**,
- For k = 2, $\Gamma_d = 1200$, $\Gamma_v = 160$, $\Gamma_t = 20$, ARI = **0.836**,
- For k = 3, $\Gamma_d = 1200$, $\Gamma_v = 160$, $\Gamma_t = 20$, ARI = **0.859**,
- For k = 4, $\Gamma_d = 1200$, $\Gamma_v = 140$, $\Gamma_t = 20$, ARI = **0.856**.

The average ARI for all folds is **0.852**.

7.5.1 Results by Group Size

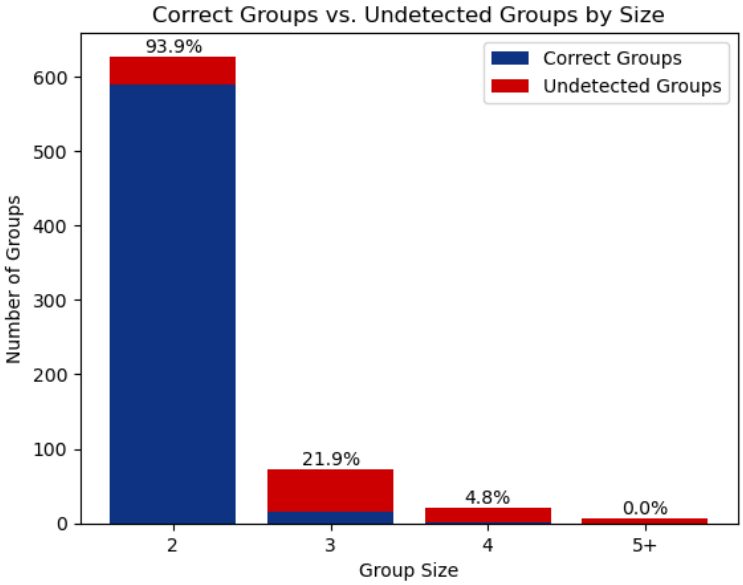

Figure 17: Distribution of ground truth groups for AHC

This time, there are more differences in the results to observe than between Time-Sequence DBSCAN and our algorithm. The result ARI of 0.852 is worse than our algorithm's (0.894) and Time-sequence DBSCAN's (0.878).

Group size	2	3	4	5+	All
Number of groups	627	73	21	7	728
Detected	589	16	1	0	606

Table 8: Number of detected ground truth groups for AHC

The AHC detected 93.9% of groups of size two, which is even better than what our algorithm was able to achieve (93.3%). Unfortunately, for all the other group sizes, the performance degraded significantly. Only 21.9% of groups of size three, and almost no detected groups of sizes beyond that (Figure 17 and Table 8).

It is not entirely clear why this is happening. One of the possible explanations could be, that the condition for graph tightness (see 2.2.1) is too strict – for an edge to exist in the graph, it is required for two people to have a common velocity vector and distance at the same time – this might turn out to be challenging for pedestrians that are walking on the opposite ends of the group, and therefore might result in algorithm prioritizing smaller groups.

False positives support this theory with a very high number of partially correct false positives. There are 179 false positives of size two, and 57 of them are size three, missing one person. Out of 17 false positives of size three, 7 are groups of size four missing one person.

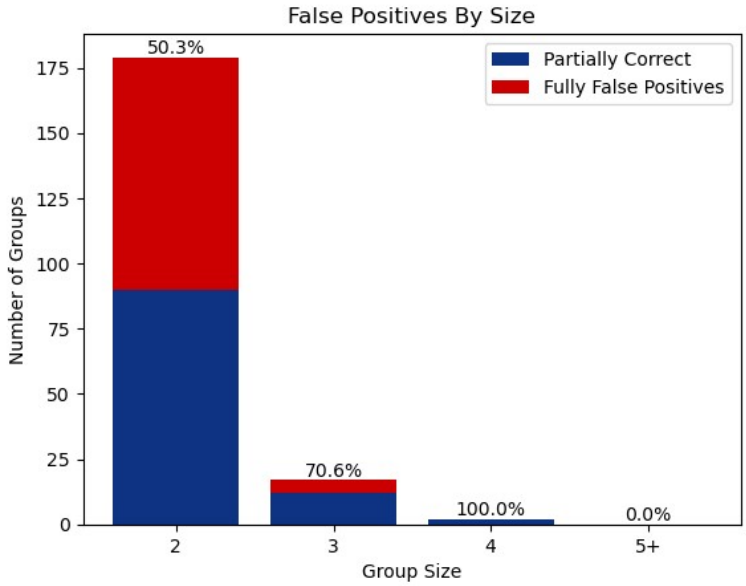

Figure 18: Distribution of false positives for AHC

Group size	2	3	4	5+	All
Number of false positives	179	17	2	0	198

Table 9: Number of false positives for AHC

7.6 Comparison of All Algorithms

Let's look at the results of all three algorithms together. Our algorithm was the most successful in detecting groups, with 648 groups from ground truth detected. Time-Sequence DBSCAN had only 9 less, and the last was AHC with 606 groups (Table 10).

Looking at the false positives, the order is exactly the same. The best was our algorithm with 118 false positives. Time-Sequence DBSCAN had 19 more and AHC performed much worse with 198 false positives (Table 11).

We are satisfied with the performance of our algorithm. The number of detected ground truth groups was very similar to the Time-Sequence DBSCAN, and the difference in score was mostly driven by having 14% fewer false positives.

Group size	2	3	4	5+	All
Our Algorithm	585	50	11	2	648
Time-Sequence DBSCAN	579	48	10	2	639
AHC	589	16	1	0	606

Table 10: Number of detected ground truth groups for each algorithm

Group size	2	3	4	5+	All
Our Algorithm	86	27	5	0	118
Time-Sequence DBSCAN	100	29	5	3	137
AHC	179	17	2	0	198

Table 11: Number of false positives for each algorithm

7.7 Investigation of Results for Day 2

We started to perform the same experiments as in previous sections on day 2 part of the dataset. During the testing, we noticed that the results were widely different, which prompted us to investigate if there might be a problem with the dataset for day 2.

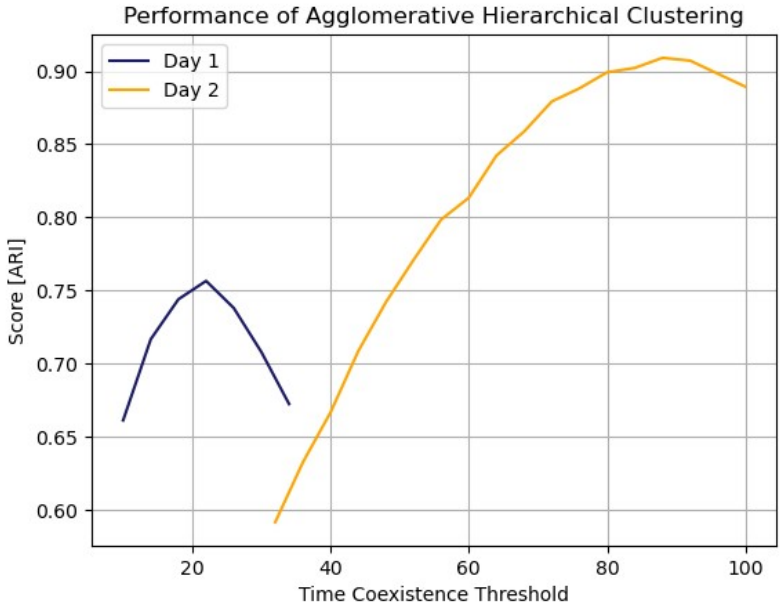

Figure 19: Performance of AHC per time coexistence threshold for day 1 and day 2, distance threshold = 2100, velocity threshold = 150

On a Figure 19, we can see the performance of the AHC algorithm when the distance (Γ_d) and velocity (Γ_v) thresholds are fixed and only the time coexistence threshold Γ_t is changing. The AHC algorithm is improving remarkably for day 2

compared to day 1 and for completely different values of time coexistence threshold. The highest ARI for this setup was around 0.92. The tests we tried on Time-Sequence DBSCAN and our algorithm had the opposite result – performance decreased. Our algorithm's ARI dropped to around 0.76. This change is beyond what we would consider normal behavior.

We tried the performance of our algorithm by taking the best parameters for both day 1 and day 2 – without proper cross-validation, a simple experiment with the whole day being used for both training and testing. For day 1 we detected 653 out of 728 groups from ground truth, and for day 2 we detected 641 out of 730 groups. This indicates that the lower performance on day 2 is not caused by the inability to detect true positives but rather by detecting too many false positives.

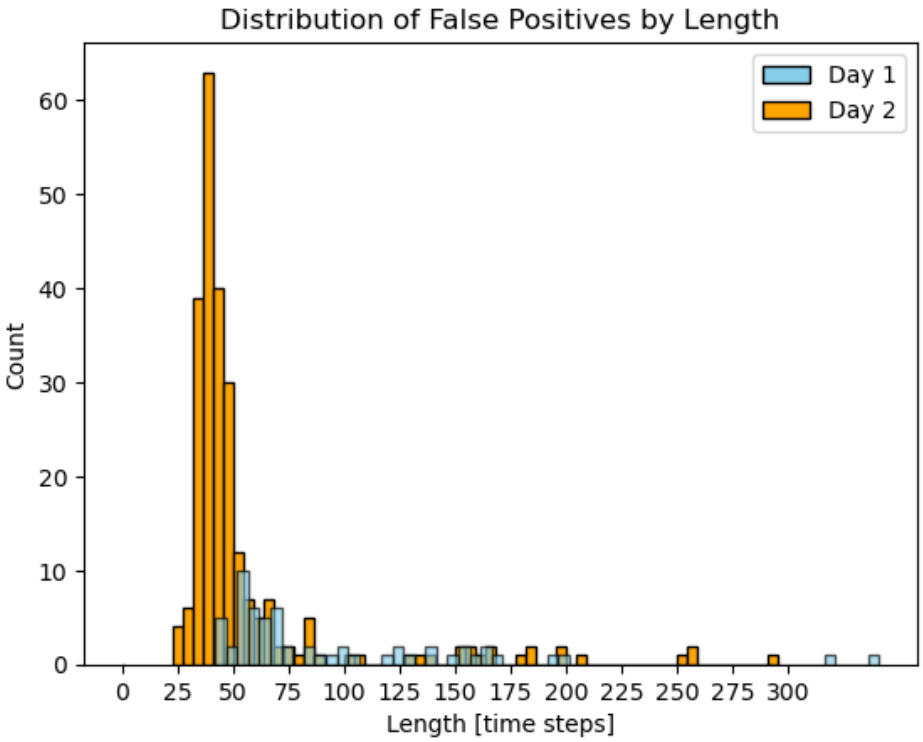

Figure 20: Distribution of false positives groups from our algorithm for day 1 and day 2

We collected false positives from both experiments and in Figure 20, we can observe a distribution of these groups by the length of the whole group (number of unique time steps the group covers). After 50 time steps, the distributions are similar. The major difference is with groups that have 50 or fewer time steps. On its own, this does not prove anything – it is possible that day 2 has many groups with shorter trajectories that we are falsely detecting.

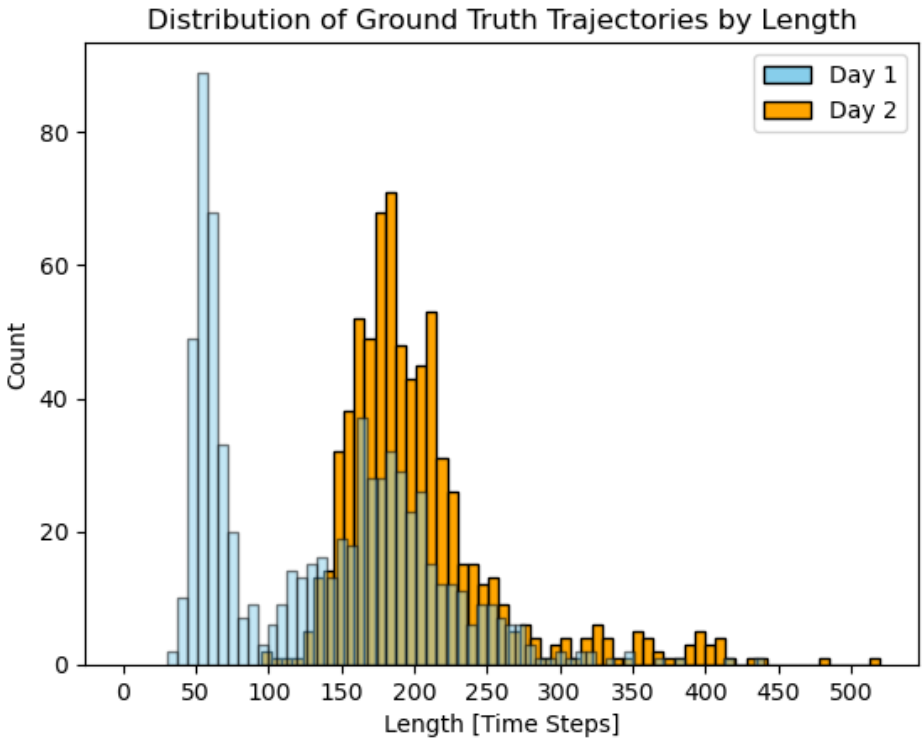

Figure 21: Distribution of ground truth groups for both days, per length

As a next step, we wanted to see what ground truth groups look like – for this, we don't need to run any algorithm. We collected all 728 and 730 ground truth groups for day 1 and day 2, respectively. Once again we can see in Figure 21 the distribution of these groups by length. We see two different distributions. Day 1 has two clearly defined peaks – one around the length of 55 and the other around the length of 175. In contrast, day 2 looks like the Gaussian distribution with a single peak around the length of 180. This is strange and should not happen when all data come from the same place, two days apart.

In the Figure 22, we displayed all trajectories from ground truth groups for day 1 and day 2, as well as false positives produced by our algorithm. All points in the figure are transparent (alpha = 0.3). From the picture for the first day, it is clear that almost all ground truth groups of length less than 100 are walking in a vertical corridor that is on the left side – this makes sense as the horizontal corridor is much longer and most likely takes more than 100 time steps (~25 seconds) to walk through it. This also explains why we have seen two peaks in the distribution for the first day – one for people crossing the vertical corridor and one for the horizontal corridor.

It is hard to believe that on the second day, there wouldn't be a single group walking in the vertical corridor with how many groups were detected by our

algorithm. The only reasonable conclusion is that the data for the second day are incorrect (incomplete), and we cannot judge accurately the results created from this day. Given these observations, for the rest of this thesis, **we will only use the first day for our experiments**.

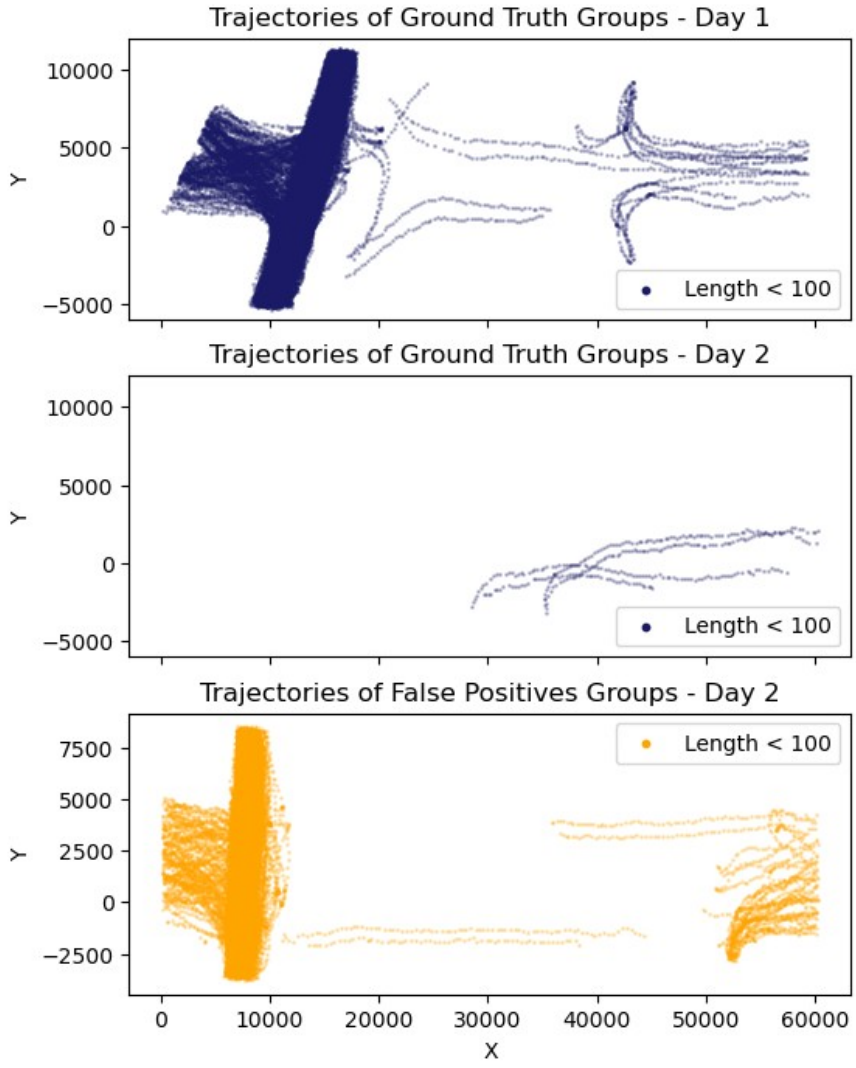

Figure 22: Trajectories with length of <100, separately for ground truth groups of day 1 and day 2, and false positives of day 2 from our algorithm

The last thing that needs explanation is why the AHC did not lose performance during testing on the second day. The opposite happened – its performance increased significantly. AHC uses graph tightness as a decider if two groups (represented by a graph) can be merged. The edge between persons i and j exists if $\rho_{ij} = \sum_t \delta_t(i,j) > \Gamma_t$ (for details see [2]). Instead of the percentage of time steps that i and j are "close" together, AHC uses a constant number. When we set $\Gamma_t = 100$, what happens is that any two trajectories shorter than 100 time steps **cannot** have an edge between each

other – and will only decrease graph tightness. This virtually removes all groups shorter than Γ_t.

For day 1, the performance started dropping around $\Gamma_t = 22$ as one would expect because short groups will never be detected. But the test on the day 2 behaved completely differently – instead of performance dropping around the same value, the performance kept increasing until $\Gamma_t = 88$. Looking back at the Figure 21, we can see that the shortest ground truth groups on day 2 are around 95 time steps long. AHC was "lucky" that it worked in this way, and it completely removed all the groups missing from the ground truth for high enough Γ_t while there were enough long groups that AHC was still able to detect.

8 Group Shape Score Validator

During the analysis of the DIAMOR dataset in 6.6.3, we noticed how well our expectations from group shape analysis aligned with the actual group shape scores of ground truth groups of sizes two and three. This led to the idea that group shape scores could be helpful as validation at the end of any algorithm that produces detected groups. This process might not help with producing groups, but removing false positives is valuable and increases the overall quality of algorithms. Validation will be done only for group sizes two and three due to the small sample size for bigger groups.

For both sizes, we will define a threshold that every group of that size needs to pass to be included in the final detected groups. This is how validation is going to work:

ALGORITHM: GROUP SHAPE SCORE VALIDATOR

Input: detected group G, group shape score thresholds thr_{gs}

Output: filtered detected groups G'

1: $G' \leftarrow \{\}$

2: **for** G_i **in** G**:**

3: size $\leftarrow |G_i|$ // if size > 3, for purposes of this thesis, add group to G'

5: **if** $thr_{gs}[\text{size}] < Gs(G_i)$:

6: G'.add(G_i)

7: **end**

8: **end**

9: **return** G'

8.1 Thresholds Selection

We used k-fold cross-validation for testing the unmodified datasets, and we will use a similar approach for testing the validator. Just like before, we will use 4 folds for all experiments.

For each fold, we will first analyze group shape scores of groups in the training part. The validator should maximize the number of removed false positives while concurrently minimizing the number of removed true positives. It seems appropriate that thresholds should be set so that most, if not all, ground truth groups from the training dataset would pass validation. The extra parameter that will be added to Time-sequence DBSCAN and AHC is *percentile.* This parameter decides values for $thr_{gs}=[thr_2, thr_3]$. For the n^{th} *percentile*, thresholds will be set to the the n^{th} percentile of all group shape scores from a training set of groups of size two and three, separately.

In the following sections, we tested different percentiles, ranging from the 0^{th} to the 7^{th} percentile.

8.2 Time-Sequence DBSCAN with Validator

We followed the same process as in previous experiments. We tested all combinations of epsilon – ranged from 1100 to 1500, examined at intervals of 50, and CTR – ranged from 0.2 to 0.95, with intervals of 0.05. Now with extra combinations for *percentile* – whole numbers ranged from 0 to 7.

The best results for each fold were as follows:

- For k = 1, percentile = 3, epsilon = 1200, CTR = 0.30, ARI = **0.925**,
- For k = 2, percentile = 6, epsilon = 1150, CTR = 0.25, ARI = **0.844**,
- For k = 3, percentile = 1, epsilon = 1450, CTR = 0.60, ARI = **0.890**,
- For k = 4, percentile = 2, epsilon = 1500, CTR = 0.50, ARI = **0.902**.

The total result for this method is **0.890**. That is an improvement from **0.878** which Time-Sequence DBSCAN achieved without validation.

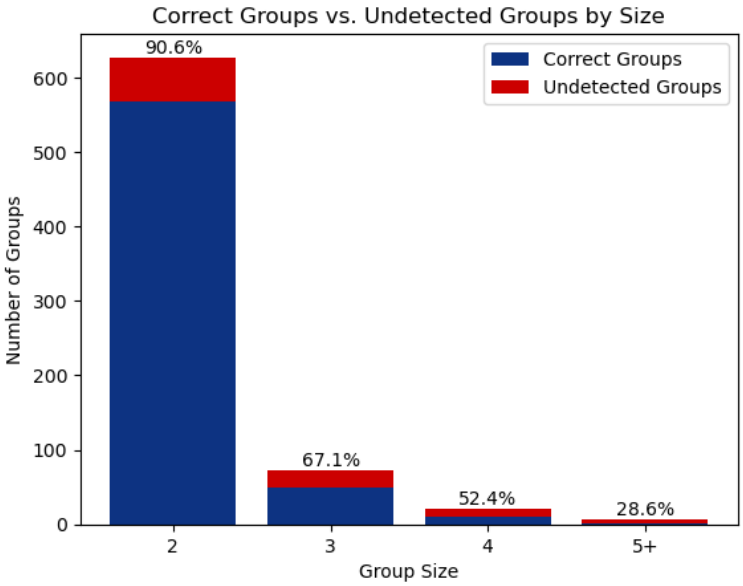

Figure 23: Distribution of ground truth groups for Time-Sequence DBSCAN

Group size	2	3	4	5+	All
Detected (original)	579	48	10	2	639
Detected (with validator)	568	49	11	2	630

Table 12: Number of detected ground truth groups for Time-Sequence DBSCAN with validator

The number of detected ground truth groups of size two decreased. This is not an unexpected result. Ground truth groups with low group shape scores will cause the validator to remove them from the detected groups. For groups of size three, we see the counterintuitive results – the number of detected ground truth groups increased by 1. This can happen due to the fact that the other constraints can be lowered (by increasing epsilon or decreasing CTR). In total, the number of detected ground truth groups with validator was lower by 9.

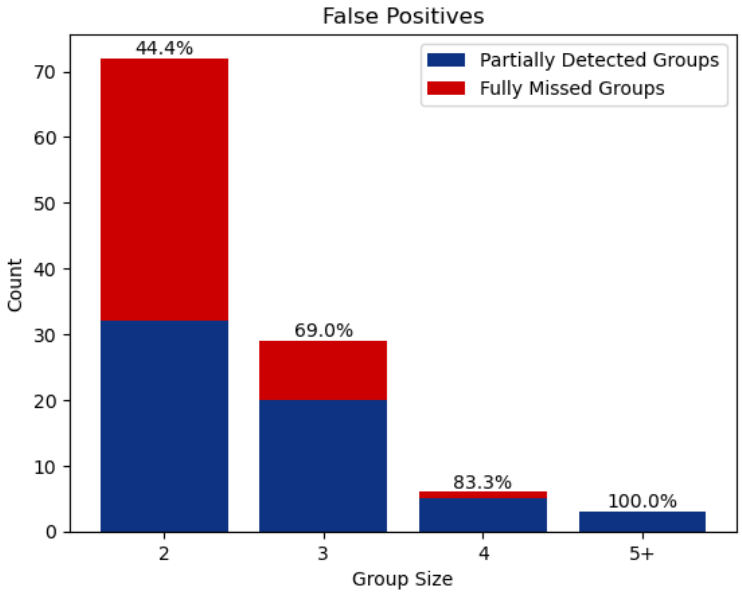

Figure 24: Distribution of false positives for Time-Sequence DBSCAN with validator

Group size	2	3	4	5+	All
Number of false positives (original)	100	29	5	3	137
Number of false positives (with validator)	72	29	6	3	110

Table 13: Number of false positives for Time-Sequence DBSCAN with validator

False positives have similar results. The number of false positives for size two lowered by 28, which we hoped to happen, and the number of false positives for size three did not change.

8.3 Agglomerative Hierarchical Clustering with Validator

We followed the same process as in previous experiments. Distance threshold Γ_d ranged from 1000 to 1600 in intervals of 100, velocity threshold Γ_v ranged from 80 to 170, in intervals of 10, and time coexistence ratio threshold Γ_t ranged from 16 to 32 in intervals of 4. Like for Time-Sequence, the *percentile* is ranging from 0 to 7.

The best results for each fold were as follows:

- For k = 1, percentile = 5, Γ_d = 1300, Γ_v = 120, Γ_t = 16, ARI = **0.881**,
- For k = 2, percentile = 4, Γ_d = 1200, Γ_v = 160, Γ_t = 20, ARI = **0.828**,
- For k = 3, percentile = 4, Γ_d = 1300, Γ_v = 130, Γ_t = 20, ARI = **0.864**,

- For k = 4, percentile = 3, Γ_d = 1300, Γ_v = 130, Γ_t = 20, ARI = **0.880.**

The result for AHC with validation is **0.863**. This is an improvement from **0.856** without validation.

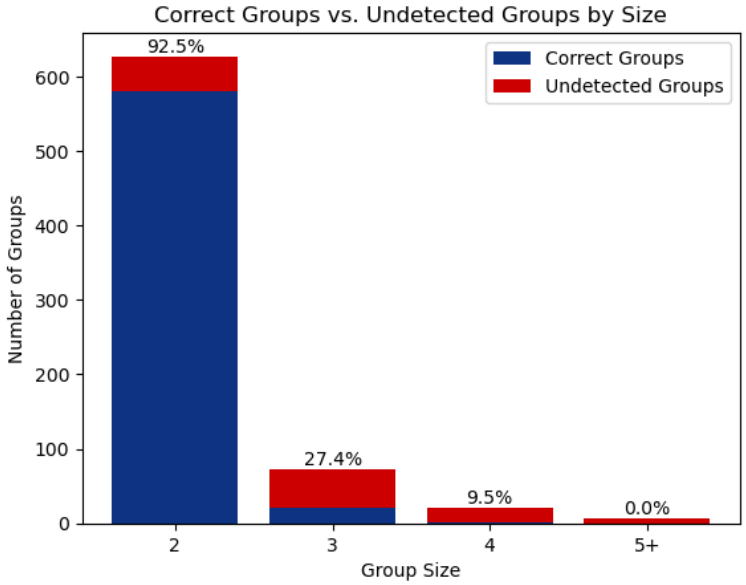

Figure 25: Distribution of ground truth groups for AHC with the validator

Group size	2	3	4	5+	All
Detected (original)	589	16	1	0	606
Detected (with validator)	580	20	2	0	602

Table 14: Number of detected ground truth groups for AHC with the validator

The number of detected groups decreased only marginally, from 606 groups without the validator to 602 with validator. We see similar effect as for Time-Sequence DBSCAN – for groups of size two there is a decrease as expected but on the other hand, for groups of size three there is an increase. We assume the reason is the same as for Time-Sequence DBSCAN.

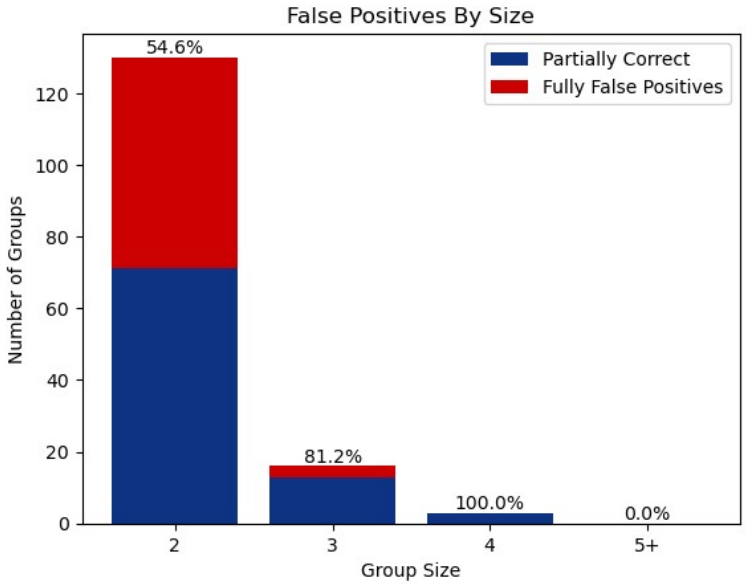

Figure 26: Distribution of false positives for AHC with the validator

Group size	2	3	4	5+	All
Number of false positives	179	17	2	0	198
Number of false positives (with validator)	130	16	3	0	149

Table 15: Number of false positives for AHC with the validator

We see a nice decrease in false positives from 198 without a validator to 149 with the validator. The number of false positives of size two is 49. For groups of size three, the difference is only 1 group.

Regrettably, the validator also removed a lot of false positives that were partially correct. Before applying validator, the AHC detected 90 partially correct groups of size two and after applying validator, there were 71. This might be the reason why ARI did improve less than we would expect. AHC's strong preference for groups of two goes against the validator because a group of size two is compared to the threshold we would expect from a group of size two. The validator makes a binary choice whether to keep the group or not. For groups of size two that are in reality groups of size three or bigger, this threshold is too strict, and therefore many get discarded in the process.

9 Dataset Modification

When running experiments, we noticed our dataset was not as robust or variable as we would like. By the nature of the task, it is hard to get a dataset that has time steps with a high density of people and also includes a ground truth that can be fully trusted – the higher the density, the harder it is for human annotators to be able to recognize social groups.

There are datasets capturing busy locations that have thousands of people moving through in minutes, but for these, it is almost impossible to truly say if people are walking together or not. It would probably require a highly controlled environment with auditors at every entry and exit point to confidently create a dataset with ground truth about social groups.

This chapter introduces a method of modifying datasets that could be beneficial when algorithms exhibit very similar performance. Ultimately, we will test our algorithms on a modified DIAMOR dataset.

9.1 Increasing Inner Group Distances

The aim of this modification is to increase the distance between pedestrians inside the groups while changing anything else as little as possible. The idea behind this modification is that if we increase the distance between pedestrians inside groups (based on the ground truth that we have) while other pedestrians are unchanged, we will somewhat simulate a higher density of a crowd.

For each group, we calculate the group member's general position relative to the group's centroid – for this, we need some approximation of the general shape of the group. Based on our testing, it turns out that the group's shape at the middle time step of the group's existence in the dataset is a good approximation for this dataset. We calculate the centroid (4.5.2) of each group at the time step we selected for that group.

For each group member, we calculate a displacement vector from the centroid to that member. The new position of each group member is given by adding the *scaled* displacement vector to the centroid. We do that for every time step – essentially

making a constant shift across the whole trajectory. Example for one time step in Figure 27.

The *scale* of the displacement vector will be the parameter we will focus on during testing. A *scale* of 1.5 represents an increase in distance from pedestrians to centroid by 50%.

Remember that the calculation of the displacement vector is done for one time step, and used for all time steps. That makes this modification a constant shift for pedestrians in groups, and therefore, this modification does not change the correlation of movement.

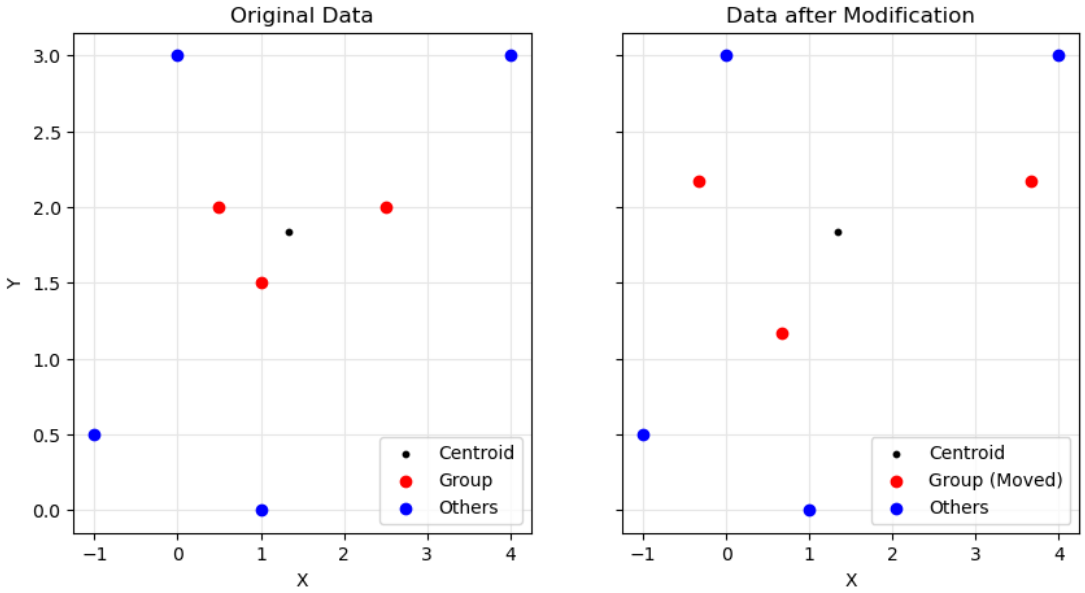

Figure 27: Example of scaling process

9.1.1 Testing Process

This time, instead of doing proper k-fold cross-validation, we will perform both training and testing on the whole dataset. We are looking for trends – how much the performance decreases with increasing inner group distance. We chose to omit the cross-validation due to time constraints. The decision to streamline the testing may limit the precision of results, but the chosen approach should be adequate to understand trends caused by our modification.

For all algorithms, we tested the performance for *scale* = 1 (no change), 1.5, 2, and 2.5, where *scale* = 1 is used as a baseline for comparison to an unchanged dataset. Previous experiments used *scale* = 1 by default with k-fold cross-validation. Please note that results without k-fold validation might differ from these. Experiments for Time-sequence DBSCAN and AHC will also include group shape

validation for each test. For a detailed analysis of results, we are going to focus purely on ***scale* = 2**, as we believe it provides significant, yet hopefully still realistic, change to the dataset.

9.2 Performance of Our Algorithm

Again, we run the evolutionary algorithm to search for the best parameters. These are the best-performing parameters for each *stretch*:

- For *scale* = 1, thr_d = 1317, thr_{sc} = 0.35, thr_{st} = 0.30, thr_{gs} = [0.32, 0.17, 0.08, 0.06], ARI = **0.902**,
- For *scale* = 1.5, thr_d = 1841, thr_{sc} = 0.27, thr_{st} = 0.29, thr_{gs} = [0.55, 0.49, 0.39, 0.34], ARI = **0.857**,
- For *scale* = 2, thr_d = 2533, thr_{sc} = 0.72, thr_{st} = 0.35, thr_{gs} = [0.56, 0.56, 0.46, 0.26], ARI = **0.811**,
- For *scale* = 2.5, thr_d = 3272, thr_{sc} = 0.68, thr_{st} = 0.52, thr_{gs} = [0.59, 0.56, 0.50, 0.31], ARI = **0.768**.

As expected, the performance is dropping with each increase of *scale* as pedestrians that are part of some group are getting closer to other pedestrians. We can observe how the algorithm was forced to use an increased distance threshold thr_d and, starting from *scale* = 1.5, how the shape thresholds thr_{gs} increased. The number of false positives was likely too high (due to pedestrians being closer to each other), and it was more beneficial to have higher thresholds that would lower the false positives than to allow more people to create groups.

Another interesting change is visible on thr_{sc}, which for *scale* = 2 and 2.5 increased from around 0.3 to around 0.7. The similarity of movement did not change, so it makes sense that cosine similarity would be more useful as the distances increased.

9.2.1 Results by Group Size

As mentioned earlier, our primary focus is *scale* = 2. When compared to *scale* = 1, the number of detected ground truth groups dropped for all group sizes except the biggest groups, which stayed the same. For groups of size two, the number of detected ground truth groups dropped from 587 to 501. For groups of size three, the

drop was from 53 to 42. The biggest drop by a percentage was for groups of size four – from 57.1% to just 19%.

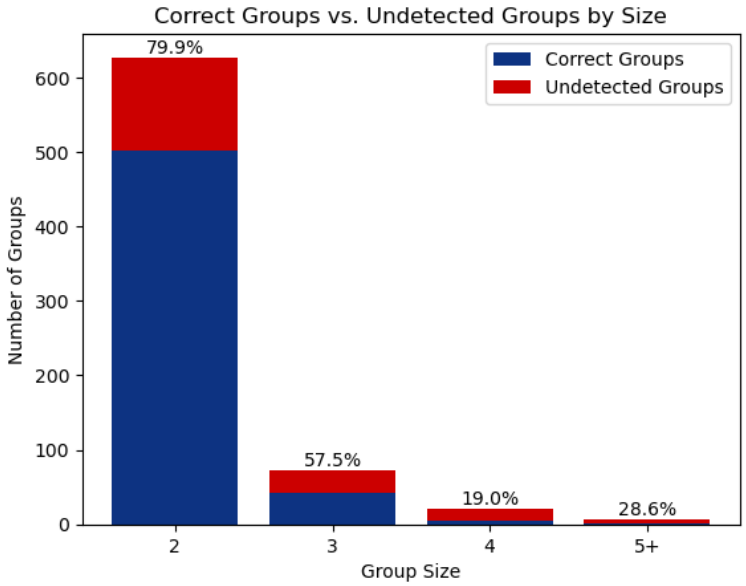

Figure 28: Distribution of ground truth groups for our algorithm, scale = 2

Group size	2	3	4	5+	All
Detected (scale = 1)	587	53	12	2	728
Detected (scale = 2)	501	42	4	2	606

Table 16: Number of detected ground truth groups per group size for our algorithm, scale comparison

The biggest difference between these two scales is in the number of false positives. For *scale* = 2, the number of false positives increased by 73%. The increase in false positives is happening for all group sizes, and even the biggest groups are now falsely detected.

Group size	2	3	4	5+	All
False positives (scale = 1)	80	28	5	0	113
False positives (scale = 2)	131	42	10	12	195

Table 17: Number of false positives per group size for our algorithm, scale comparison

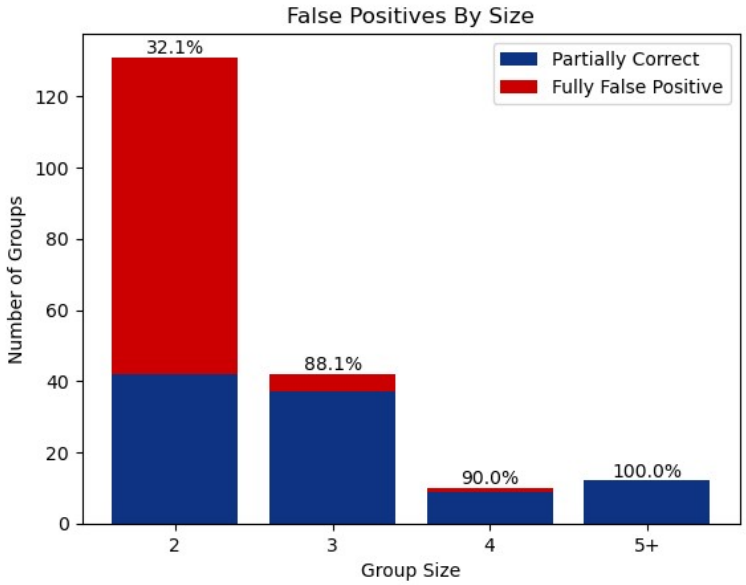

Figure 29: Distribution of false positives for our algorithm, scale = 2

9.3 Performance of Time-Sequence DBSCAN

The epsilon parameter of Time-Sequence DBSCAN was tested in different ranges for each *scale*. The tested ranges were, in our opinion, adequate, and lower or higher values did not provide any additional benefit. Epsilon was tested in increments of 100. The range for CTR was the same as in previous experiments – range between 0.2 and 0.95, with intervals of 0.05. For tests with group shape validation, the *percentile* ranged from 0 to 10, whole numbers only.

The best results for each *scale*:

- For *scale* = 1, epsilon = 1200, CTR = 0.30, ARI = **0.886**,
- For *scale* = 1.5, epsilon = 1700, CTR = 0.60, ARI = **0.824**,
- For *scale* = 2, epsilon = 2700, CTR = 0.80, ARI = **0.787**,
- For *scale* = 2.5, epsilon = 3400, CTR = 0.85, ARI = **0.749.**

And with group shape score validation:

- For *scale* = 1, percentile = 2, epsilon = 1200, CTR = 0.30, ARI = **0.899**,
- For *scale* = 1.5, percentile = 5, epsilon = 1700, CTR = 0.50, ARI = **0.845**,
- For *scale* = 2, percentile = 7, epsilon = 2700, CTR = 0.65, ARI = **0.811**,
- For *scale* = 2.5, percentile = 7, epsilon = 3700, CTR = 0.80, ARI = **0.792**.

We see similar results as for our algorithm. The performance drops with each increase in *scale,* and epsilon is forced to increase. The CTR increases with *scale* as it tries to offset the higher epsilon. The same happens with percentile, which removes more false positives with a larger *scale.*

With the validator, the performance for *scale* = 2 was the same as the performance of our algorithm, and for *scale* = 2.5, it even surpassed our algorithm (0.792 vs. 0.768).

9.3.1 Comparison of Results With and Without Validator

Our primary interest lies in how the performance changed between Time-Sequence with and without the group shape score validator. These are the results for *scale* = 2.

Group size	2	3	4	5+	All
Detected	510	41	7	2	560
Detected (with validator)	488	45	8	3	544

Table 18: Number of detected ground truth groups for Time-Sequence DBSCAN, scale = 2

For groups of size two, without the validator, Time-Sequence DBSCAN detected 71% more false positives than with the validator. For other sizes, the false positives increased. With the validator, we have 16 more false positives of size three. This happens because CTR declined from 0.8 without a validator to 0.6 with a validator. Validator provides a way how to remove wrongly detected groups, and in response, the CTR can be lowered to allow more groups to be detected.

Group size	2	3	4	5+	All
Number of false positives	142	36	9	4	191
Number of false positives (with validator)	83	52	16	15	166

Table 19: Number of false positives per group size for Time-Sequence DBSCAN, scale = 2

9.4 Performance of Agglomerative Hierarchical Clustering

Now, let's examine the performance of AHC. The AHC algorithm has three parameters. As mentioned earlier, we are interested in trends caused by our modification. To streamline the testing, we locked the Γ_t parameter to 22. The

distance threshold was tested in ranges appropriate for the *scale,* in intervals of 100, and the velocity threshold in a range from 60 to 150, in intervals of 10. For tests with group shape validation, the *percentile* ranged from 0 to 10, whole numbers only.

The best results for each *scale*:

- For *scale* = 1, Γ_d = 1300, Γ_v = 130, ARI = **0.860**,
- For *scale* = 1.5, Γ_d = 1700, Γ_v = 110, ARI = **0.770**,
- For *scale* = 2, Γ_d = 2100, Γ_v = 100, ARI = **0.671**,
- For *scale* = 2.5, Γ_d = 2800, Γ_v = 80, ARI = **0.588.**

And with group shape score validation:

- For *scale* = 1, percentile = 2, Γ_d = 1300, Γ_v = 130, ARI = **0.869**,
- For *scale* = 1.5, percentile = 5, Γ_d = 1700, Γ_v = 120, ARI = **0.800**,
- For *scale* = 2, percentile = 7, Γ_d = 2700, Γ_v = 120, ARI = **0.757**,
- For *scale* = 2.5, percentile = 7, Γ_d = 3300, Γ_v = 100, ARI = **0.722.**

The AHC performance decreases with *scale* the most. For original data, the ARI is 0.860, which goes down to 0.588 for *scale* = 2.5. The validator seems to be providing a significant boost. For *scale* = 2 ARI improves from 0.671 to 0.757 and for *scale* = 2.5 ARI improves from 0.588 to 0.722.

9.4.1 Comparison of Results With and Without Validator

Once again, we will highlight the performance changes between AHC with and without the group shape score validator. These are the results for *scale* = 2.

Group size	2	3	4	5+	All
Detected	486	3	0	0	489
Detected (with validator)	481	12	0	0	493

Table 20: Number of detected ground truth groups for AHC, scale = 2

Like in previous experiments on AHC, we see that it was relatively effective in detecting groups of size two and ineffective for any other sizes. The number of detected groups from ground truth is almost the same. The decreased number of false

positives is the main factor in increased ARI – from 311 without the validator to 214 with the validator.

Group size	2	3	4	5+	All
Number of false positives	287	19	5	0	311
Number of false positives (with validator)	140	54	16	4	214

Table 21: Number of false positives per group size for AHC, scale = 2

9.5 Summary of All Algorithms

Our algorithm had the best ARI when compared to the other two algorithms without the group shape score validator. With the validator, Time-Sequence DBSCAN had the same ARI. It seems like our validator could be useful when applied to algorithms that are tested on datasets with lower distances between pedestrians, but more testing needs to be done.

In the following tables, you will find a summary of all algorithms and their results for *scale* = 2.

		ARI
Our Algorithm		0.811
Time-Sequence DBSCAN	original	0.787
	with validator	0.811
AHC	original	0.671
	with validator	0.757

Table 22: ARI for all algorithms, scale = 2

Detection of ground truth groups:

Group size		2	3	4	5+	All
Our Algorithm		501	42	4	2	549
Time-Sequence DBSCAN	original	510	41	7	2	560
	with validator	488	45	8	3	544
AHC	original	486	3	0	0	489
	with validator	481	12	0	0	493

Table 23: Number of detected ground truth groups per group size for all algorithms, scale = 2

False positives:

Group size		2	3	4	5+	All
Our Algorithm		131	42	10	12	195
Time-Sequence DBSCAN	original	142	36	9	4	191
	with validator	83	52	16	15	166
AHC	original	287	19	5	0	311
	with validator	140	54	16	4	214

Table 24: Number of false positives per group size for all algorithms, scale = 2

10 Conclusion

In this thesis, we systematically analyzed group crowd behavior, focusing on characteristics like group shape, distance, and size from a sociological angle. Drawing upon these insights, we developed an algorithm dedicated to solving the group detection problem. This proposed solution introduces a group shape score, a measure used to evaluate a group's shape according to positioning and movement direction, and employs this measure as a decisive factor in determining the validity of groups.

The evaluation process involved a comparative analysis against two existing solutions — Time-sequence DBSCAN [1] and Agglomerative Hierarchical Clustering with Hausdorff Distance [2]. We utilized the DIAMOR dataset [3] for this comparison. In this testing, our algorithm was consistently better, which was mainly driven by avoidance of false positives, having 14% fewer false positives than Time-sequence DBSCAN and 40% fewer than Agglomerative Hierarchical Clustering.

We also introduced a validator as a possible addition to post-processing methods in group detection algorithms. This validator uses a similar approach to our proposed algorithm in scoring group shape and using it as a measure to verify existing group classifications. Its main contribution is eradicating false positive classifications from the results. Through testing, the validator demonstrated that it could improve the performance of both Time-sequence DBSCAN and Agglomerative Hierarchical Clustering on the DIAMOR dataset.

In order to properly test all aspects of proposed algorithm and validator, it would be beneficial to have a robust dataset exhibiting wide range of complex patterns for training and testing purposes. One example of such pattern is higher inner group distance. Regrettably, to create or obtain such a dataset is truly challenging. Due to this, we introduced a dataset modification in order to harden the conditions for group detection – widening distances between members of the group. As a result, this created a bigger gap between the performance of the algorithms. Our group shape validator has proven effective when used on a modified dataset improving both Time-DBSCAN and Agglomerative Hierarchical Clustering on the DIAMOR

dataset. More dataset modifications could be created and tested. This represents wide and open area for reaseach.

In future work, we would like to explore using data featuring higher-density situations or more obstacles, offering opportunities to observe different group shapes. Furthermore, there is a potential to enhance our algorithm by considering the incorporation of visual information like video footage and exploring aspects of human interactions such as hand gestures, head movements, or eye contact.

At the end of this thesis, we express the hope that our work contributes valuable insights and offers potential avenues for improvement within the field.

Bibliography

[1] H. Cheng, Y. Li and M. Sester, "Pedestrian Group Detection in Shared Space," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 1707-1714.

[2] W. Ge, R. T. Collins and R. B. Ruback, "Vision-Based Analysis of Small Groups in Pedestrian Crowds," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 34, no. 5, pp. 1003-1016, 2012.

[3] F. Zanlungo, T. Ikeda and K. Takayuki, "Dataset: Pedestrian tracking with group annotations," 2015. [Online]. Available: https://dil.atr.jp/ISL/sets/groups/. [Accessed 6 2023].

[4] J. Wąs and K. Kulakowski, Social Groups in Crowd, 2014.

[5] H. E. Driver and A. L. Kroeber, Quantitative Expression of Cultural Relationship, 1932.

[6] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification,* vol. 2, no. 1, 1985.

[7] M. Ester, H. Kriegel, J. Sander and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Knowledge Discovery and Data Mining*, 1996.

[8] R. A. Meyers, Encyclopedia of Physical Science and Technology, San Diego: Academic Press, 2002.

[9] M.-P. Dubuisson and A. Jain, "A modified Hausdorff distance for object matching," in *Proceedings of 12th International Conference on Pattern Recognition*, Jerusalem, 1994, pp. 566-568.

[10] M. Moussaïd, N. Perozo, S. Garnier, D. Helbing and G. Theraulaz, "The Walking Behaviour of Pedestrian Social Groups and Its Impact on Crowd Dynamics," *PLOS ONE,* vol. 5, pp. 1-7, 2010.

[11] A. Seyfried, B. Steffen, W. Klingsch and M. Boltes, "The fundamental diagram of pedestrian movement revisited," *Journal of Statistical Mechanics: Theory and Experiment,* 2005.

[12] M. F. Mohamad Ali, M. S. Abustan, S. Abu Talib, I. Abustan, N. Abd Rahman and H. Gotoh, "Psychological distance of pedestrian at the bus terminal area," *E3S Web of Conferences,* vol. 34, 2018.

[13] K. Katevas, H. Haddadi, L. N. Tokarchuk and R. G. Clegg, "Walking in Sync: Two is Company, Three's a Crowd," in *Proceedings of the 2nd workshop on Workshop on Physical Analytics*, 2015.

[14] R. Shepherd, C. Clegg and M. Robinson, Understanding Crowd Behaviours, Volume 1: Practical Guidance and Lessons Identified, 2010.

[15] T. L. Chartrand and J. A. Bargh, "The chameleon effect: The perception–behavior link and social interaction," *Journal of Personality and Social Psychology,* vol. 76, no. 6, pp. 893-910, 1999.

[16] C. Syms, "Principal Component Analysis," in *Encyclopedia of Ecology*, Academic Press, 2008, pp. 2940-2949.

[17] A. F. Alkarkhi and W. . A. Alqaraghuli, Easy Statistics for Food Science with R, Academic Press, 2019.

[18] H. Abdi and L. J. Williams, “Principal component analysis,” *WIREs Computational Statistics,* vol. 2, no. 4, pp. 433-459, 2010.
[19] S. Holland, Principal Components Analysis, 2008.
[20] T. Cormen, C. Leiserson, R. Rivest and C. Stein, Introduction to Algorithms (Fourth Edition), The MIT Press, 2022.
[21] F. Hutter, L. Kotthoff and J. Vanschoren, “Hyperparameter optimization,” in *AutoML: Methods, Systems, Challenges*, Springer, 2019, pp. 3-38.
[22] D. Soper, “Greed Is Good: Rapid Hyperparameter Optimization and Model Selection Using Greedy k-Fold Cross Validation,” *Electronics,* vol. 10, no. 16, 2021.
[23] D. Câmara, “Evolution and Evolutionary Algorithms,” in *Bio-Inspired Networking*, Elsevier Science, 2015.
[24] G. Neumann, “Artificial Intelligence Programming,” in *Encyclopedia of Information Systems*, Academic Press, 2003, pp. 31-45.
[25] P. Refaeilzadeh, L. Tang and H. Liu, “Cross-Validation,” in *Encyclopedia of Database Systems* , Springer US, 2009, pp. 532-538.
[26] S. Raschka, “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning,” 2020.
[27] 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, 2014.
[28] M. Federici, A. Gorrini, L. Manenti and G. Vizzari, Data Collection for Modeling and Simulation: Case Study at the University of Milan-Bicocca, 2012.
[29] Z. Michalewicz and M. Schoenauer, “Evolutionary Algorithms,” in *Encyclopedia of Information Systems*, Academic Press, 2003, pp. 259-267.
[30] G. Syswerda, “Uniform Crossover in Genetic Algorithms,” in *Proceedings of the 3rd International Conference on Genetic Algorithms*, San Francisco, 1989.
[31] O. Bell, “Applications of Gaussian Mutation for Self Adaptation in Evolutionary Genetic Algorithms,” 2022. [Online]. Available: https://arxiv.org/abs/2201.00285.
[32] C. McPhail and R. Wohlstein, “Using Film to Analyze Pedestrian Behavior,” *Sociological Methods & Research,* vol. 10, no. 3, pp. 347-375, 1982.

List of Abbreviations

AHC – Agglomerative Hierarchical Clustering

ARI – Adjusted Rand Index

BFS – Breadth-First Search

CTR – Coexisting Time Ratio

DBSCAN – Density-Based Spatial Clustering of Applications with Noise

EA – Evolutionary Algorithm

PCA – Principal Component Analysis